МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ ГОРОДСКОГО ХОЗЯЙСТВА ИМЕНИ А. Н. БЕКЕТОВА

Б. И. Погребняк, А. Б. Костенко, М. В. Булаенко

ИНФОРМАТИКА

КОНСПЕКТ ЛЕКЦИЙ

(для студентов 1-го курса дневной и заочной форм обучения образовательно-квалификационного уровня бакалавр, направлений подготовки 6.030504 – Экономика предприятия и 6.030509 – Учет и аудит)

ХАРЬКОВ

ХНУГХ им. А. Н. Бекетова

2015

Погребняк Б. И. Информатика: Конспект лекций (для студентов 1-го курса дневной и заочной форм обучения образовательно-квалификационного уровня бакалавр, направлений подготовки 6.030504 – Экономика предприятия и 6.030509 – Учет и аудит) / Б. И. Погребняк, **А. Б. Костенко, М. В. Булаенко**; Харьков нац. ун-т гор. хоз-ва им. А. Н. Бекетова. – Харьков : ХНУГХ им. А. Н. Бекетова, 2015. – 261 с.

Авторы: Б. И. Погребняк,
А. Б. Костенко,
М. В. Булаенко

Конспект лекций построен в соответствии с требованиями кредитно-модульной системы организации учебного процесса и согласован с ориентировочной структурой содержания учебной дисциплины, рекомендованной Европейской Кредитно-Трансферной Системой (ECTS).

Рекомендовано для студентов направлений подготовки 6.030504 – Экономика предприятия и 6.030509 – Учет и аудит

Рецензент: зав. кафедры Прикладной математики и информационных технологий Харьковского национального университета городского хозяйства имени А. Н. Бекетова д.т.н., проф. Н. И. Самойленко.

Утверждено на заседании кафедры Прикладной математики и информационных технологий.

Протокол № 1 от 30 августа 2014 г.

ОГЛАВЛЕНИЕ

ЛЕКЦИЯ № 1

ПРЕДМЕТ, ЦЕЛЬ И ЗАДАЧИ ДИСЦИПЛИНЫ

План

1. Основной инструмент обработки экономической информации
2. Неотъемлемая часть программного обеспечения офиса
3. Основное внимание в дисциплине
4. Предмет изучения
5. Цель изучения
6. Задачи изучения

1. Основной инструмент обработки экономической информации

Наиболее распространенным инструментом обработки информации специалиста по экономике является персональный компьютер. Он является основным оборудованием экономических и финансовых подразделений предприятий любой сферы деятельности. С его помощью выполняется подавляющее большинство операций по обработке экономической информации.

2. Неотъемлемая часть программного обеспечения офиса

Несмотря на стремительное, часто даже не поддающееся предсказанию, развитие компьютерных средств и информационных технологий, неотъемлемой частью программного обеспечения любого офиса, как сегодня, так и в ближайшем будущем, являются интегрированные офисные пакеты общего назначения, а также клиентские программы для работы в Интернете. Так происходит потому, что большинство руководителей экономических служб предприятий не считают целесообразным самостоятельно разрабатывать новые компьютерные программы, а предпочитают пользоваться уже готовыми.

Вместе с тем современные специалисты управления (экономисты, бухгалтеры, менеджеры) очень тесно взаимодействуют со специалистами в области информационных технологий. Поэтому важным фактором эффективности такого взаимодействия является владение специалистами управления основной компьютерной терминологией, понимание реальных возможностей и особенностей применения информационных технологий, знание тенденций их развития и совершенствования, умение четко сформулировать свои требования как пользователя к подобным компьютерным системам.

3. Основное внимание в дисциплине

Поэтому в дальнейшем основное внимание будет уделяться именно этим аспектам, а не особенностям построения и функционирования отдельных компонентов информационных систем. Изложение теоретических аспектов создания и функционирования таких систем будет сокращено до минимума - будут оставлены лишь базовые, фундаментальные положения, без знания которых не возможно понимание специфики их построения и функционирования. Для профессионала в области экономики изучение дисциплины «Информатика» носит прикладной характер, т.е. опираясь на полученные знания по информатике, он должен уметь эффективно решать свои основные функциональные задачи, а в случае необходимости, так же принимать обоснованные решения по вопросам совершенствования и дальнейшего развития информационных систем.

4. Предмет изучения

Предметом изучения дисциплины «Информатика» являются *средства автоматизации информационных процессов с использованием экономических данных*.

5. Цель изучения

***Цель**: формирование знаний о принципах построения и функционирования вычислительных машин, организации вычислительных процессов на персональных компьютерах и их алгоритмизации, программном обеспечении персональных компьютеров и компьютерных сетей, а также эффективном использовании современных информационно-коммуникационных технологий в профессиональной деятельности.*

6. Задачи изучения

***Задачи**: изучение теоретических основ информатики и приобретение навыков использования прикладных систем обработки экономических данных и систем программирования для персональных компьютеров и локальных компьютерных сетей во время исследования социально-экономических систем и решения задач профессионального направления.*

ЛЕКЦИЯ № 2

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

План

1. Информатика как наука
2. Источники информатики
3. Общая схема обработки информации на компьютере
4. Понятие информации
5. Методы получения информации
6. Свойства информации
7. Единицы измерения информации
8. Сигналы, сообщения, данные
9. Передача информации
10. Формы представления информации
11. Экономическая информация

11.1. Понятие экономической информации

11.2. Классификация экономической информации

11.3. Свойства экономической информации

11.4. Структура экономической информации

1. Информатика как наука

Информатика – это наука о способах и методах представления, обработки, передачи и хранения информации с помощью компьютера. Сам термин «информатика» происходит от французского Informatiqe. Впервые он возник в 60-е годы XX столетия путем слияния двух слов Informacion (Информация) и Automatique (Автоматика) для обозначения деятельности по автоматизированной обработке информации с помощью ЭВМ. Кроме Франции термин информатика широкое распространение получил также в большинстве стран Европы. В тоже время в англоязычных странах аналогичный круг

проблем обозначается термином «Computer Science», что означает буквально «Компьютерная наука».

Международный конгресс по информатике 1978 г. в Японии предложил следующее определение: «Понятие информатики охватывает области, связанные с разработкой, созданием, использованием и материально-техническим обслуживанием систем обработки информации, включая машины, оборудование, математическое обеспечение, организационные аспекты, а также комплекс промышленного, коммерческого, административного, социального и политического воздействия». В 1982 г. бывшем СССР выходит монография академика В. М. Глушкова «Основы безбумажной информатики». А год спустя годичное Общее собрание Академии наук СССР принимает решение о создании отделения «Информатики, вычислительной техники и автоматизации». Тогда же информатика стала трактоваться как «комплексная научная и инженерная дисциплина, изучающая все аспекты разработки, проектирования, создания, оценки, функционирования основанных на ЭВМ систем переработки информации, их применения и воздействия на различные области социальной практики».

2. Источники информатики

Информатика как наука сложилась сравнительно недавно. Ее развитие самым тесным образом связано с появлением в середине XX века электронных вычислительных машин (ЭВМ), которые являются универсальным средством для хранения, обработки и передачи информации. В качестве источников информатики обычно называют две науки – документалистику и кибернетику.

Основным направлением документалистики является изучение рациональных средств и методов повышения эффективности документооборота. Она сформировалась в конце XIX века в связи с бурным развитием производственных отношений, а расцвет ее пришелся на 30-е годы XX века.

Кибернетика, как наука об общих закономерностях управления и связи в различных системах – технических, социальных, биологических, возникла и начала интенсивно развиваться после второй мировой войны. Создателем кибернетики принято считать американского ученого Норберта Винера, опубликовавшего в 1948 г. свою знаменитую книгу «Кибернетика, или управление и связь в животном и машине». В ней были показаны пути создания общей теории управления и заложены основы методов рассмотрения проблем управления и связи для различных систем с единой точки зрения. Развиваясь одновременно с развитием электронно-вычислительных машин, кибернетика со временем превратилась в более общую науку о преобразовании информации. Сам же термин «кибернетика» происходит от греческого слова kyberneticos – искусный в управлении.

3. Общая схема обработки информации на компьютере

Обработка информации с помощью компьютера выполняется точно так же, как фарш в мясорубке. То есть, в мясорубку закладываются ломтики мяса, а после обработки получается тоже мясо, только перекрученное – фарш. Аналогично, на вход компьютера подают исходную информацию, а на выходе получают уже обработанную – выходную. Тогда схема обработки информации на компьютере в самом общем виде может быть представлена следующим образом (рисунок 2.1). Из этого рисунка видно, что в этом процессе участвуют три составляющие:

1) информация,
2) компьютер и
3) программы.

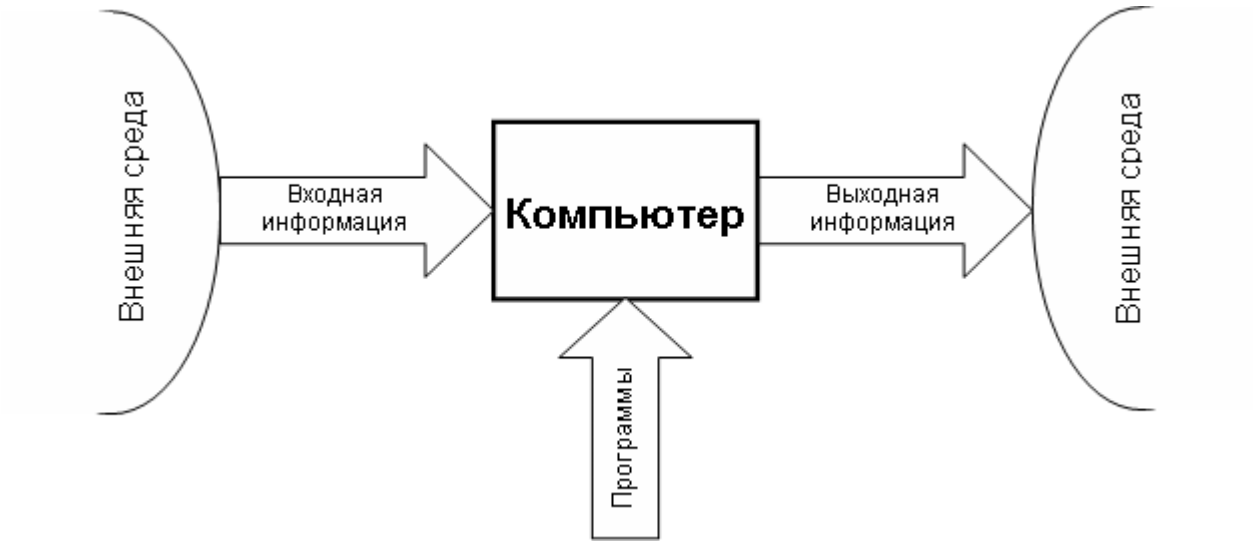


Рисунок 2.1 – Общая схема обработки информации на компьютере

4. Понятие информации

Термин «информация» происходит от латинского слова «informatio», что означает сведения, разъяснения, изложение, осведомленность, знания и т.д. Само понятие информация является одним из фундаментальных в современной науке вообще, и базовым для информатики в частности. Наряду с такими понятиями, как вещество, энергия, пространство и время, оно рассматривается в качестве важнейшей сущности мира, в котором мы живем. Его нельзя определить через более простые понятия. В математике, например, аналогичными «неопределяемыми» понятиями являются «точка» или «прямая», относительно которых можно сделать некоторые утверждения, но сами они не могут быть определены с помощью более элементарных понятий. Понятие «информация» имеет различный смысл в экономике, науке, технике, различных житейских ситуациях.

В простейшем бытовом значении под информацией понимают сведения об окружающем нас мире и протекающих в нем процессах, воспринимаемые человеком или специальными устройствами. Под информацией в технике понимают сообщения, передаваемые в форме знаков или сигналов. Под информацией в семантическом (смысловом) аспекте понимают сведения, обладающие некоторой новизной. Под информацией в кибернетике, по

определению Норберта Винера понимают ту часть знаний, которая используется для ориентирования, активного действия, управления, т.е. в целях сохранения, совершенствования, развития системы. В середине XX века слово «информация» в узком техническом смысле ввел Клод Шеннон применительно к теории связи и передачи сигналов, которая получила название «Теория информации». Под информацией в ней понимают не любые сведения, а лишь те которые, снимают полностью или уменьшают существующую неопределенность относительно некоторого события. То есть, *информация – это мера снятия неопределенности в отношении исхода интересующего нас события*.

Применительно к компьютерной обработке под информацией понимают:

1) *некоторую последовательность знаков (кодов, сигналов),*
2) *которая несет некоторую смысловую нагрузку*
3) *и представлена в понятном для компьютера виде.*

5. Методы получения информации

Информация о любом событии, объекте, его параметрах, и т.д. может быть получена одним из следующих методов:

1. *Опыт*. Является важнейшим методом получения информации. Он состоит в том, что ежедневно каждый человек, вольно, или невольно, накапливает определенное количество информации – некоторый опыт. В прошлом этот метод был основным и единственным для получения информации в жизни и развитии человека. Множество замечательных достижений было получено опытным путем, в процессе накопления опыта и выводом определенных умозаключений. Например, какого совершенства достигли древние мастера Китая в изготовлении фарфора, тульские оружейники – в изготовлении оружия.
2. *Эвристический подход*. Его название происходит от слова «эврика», что в переводе с древнегреческого означает «Я нашел!». При эвристическом

подходе проводят многократные эксперименты, после которых отбирают наиболее удачные варианты. Поэтому он еще называется «Методом проб и ошибок». Однако этот метод довольно длительный и трудоемкий, а потому недостаточно эффективный.

3. *Целенаправленный поиск*. Он характеризуется тем, что производится не беспорядочный перебор всех возможных вариантов, а анализируются известные достижения в конкретной области, планируются и проводятся опыты. В результате применения целенаправленного поиска человечество создало новые материалы и процессы, ранее не известные в природе. Этому способу получения информации способствует развитие современной техники, которая позволяет обрабатывать огромные объемы информации и получать при этом все новые и новые результаты.

Информация может быть получена так же:

1. путем наблюдения за объектом,
2. эксперимента над ним или
3. путем логического вывода.

Поэтому информация бывает:

1. доопытная (или априорная) и
2. послеопытная, (или апостериорная), полученная в результате проведенного эксперимента.

6. Свойства информации

Информация обладает множеством различных свойств. Наиболее существенными из них являются следующие:

1. **Объективность**. Информация объективна, если она не зависит от чьего-либо мнения.
2. **Достоверность**. Информация достоверна, если она отражает истинное положение дел.
3. **Полнота**. Информацию можно считать полной, если ее достаточно для понимания и принятия решения.

4. **Актуальность** – важность, существенность для настоящего времени.
5. **Адекватность** – определенный уровень соответствия создаваемого с помощью полученной информации образа реальному объекту, процессу, явлению.

7. Единицы измерения информации

С точки зрения компьютерной обработки наиболее важным свойством информации является ее количество. Определение минимальной единицы измерения количества информации связано с ее вероятностным понятием. Так, если вероятность свершения, или несвершения, какого-либо события одинакова, т.е. равна 0,5 (например, при подбрасывании монеты), то для передачи информации о нем достаточно одного двоичного разряда. Эта единица измерения была названа *бит* (от английских слов binary digital – двоичный разряд). *Один бит – это количество информации необходимое для различения двух равновероятных событий* (**1 бит** = {0, 1}). Бит также хорошо сочетается с особенностями построения современных компьютеров, поскольку они также работают по принципу «включено – выключено», «есть сигнал – нет сигнала», «высокий уровень – низкий уровень», «заряжен – разряжен» и т.д.

Количество информации равное 8-ми битам называется *байт* (**1 байт** = 8 бит). При помощи 8-ми двоичных разрядов можно записать 256 (2^8) различных значений от 00000000 до 11111111.

На практике для облегчения работы с большими объемами информации применяют более крупные единицы измерения, такие, как:

1 Килобайт (Кбайт) = 1024 байт = 2^{10};

1 Мегабайт (Мбайт) = 1024 Кбайт = 2^{20};

1 Гигабайт (Гбайт) = 1024 Мбайт = 2^{30};

1 Терабайт (Тбайт) = 1024 Гбайт = 2^{40};

1 Петабайт (Пбайт) = 1024 Тбайт = 2^{50}.

8. Сигналы, сообщения, данные

Основной особенностью информации является то, что она является категорией нематериальной. Следовательно, для существования и распространения в материальном мире информация должна быть обязательно связана с какой-либо материальной основой – без нее информация не может проявиться, передаваться и сохраняться. Поэтому *материальный объект или среду, которые служат для представления и передачи информации, называют* ***материальным носителем***. Материальным носителем информации может быть бумага, воздух, лазерный диск, электромагнитное поле и пр. *Изменение некоторой характеристики материального носителя, которое используется для представления информации, называют* ***сигналом***. *Значение этой характеристики, отнесенное к некоторой шкале измерений, называется* ***параметром сигнала***. Например, при разговоре материальным носителем является воздух, сигналом – звуковая волна, которая в нем распространяется, а параметрами сигнала – высота и громкость звука.

Однако одиночный сигнал не может содержать много информации. Поэтому для передачи информации используется ряд следующих друг за другом сигналов. Такая *последовательность сигналов называется* ***сообщением***. Примерами сообщений являются музыкальное произведение, телепередача, команды регулировщика на перекрестке, текст, распечатанный на принтере, данные, полученные в результате работы программы и т.д. *Зарегистрированные каким-либо образом на конкретном материальном носителе сигналы называются* ***данными***. Сам термин «данные» происходит от латинского слова «data» – факт, в то время как «информация» – от латинского «informatio» – сведения, разъяснения, изложение и т.д. Можно сказать, что данные выступает в качестве материальной основы для представления информации. Другими словами, данные служит «исходным сырьем» для получения информации.

Отсюда следует важный вывод о том, что одни и те же данные могут нести различную информацию для различных потребителей. Так, например,

данные об антропологических параметрах строения человека для портного, врача и спортивного тренера несут различную информацию. Для портного – это количество необходимого материала для пошива одежды и особенности его раскроя, для врача – отклонения от нормы в пропорциях фигуры и возможные патологии, а для спортивного тренера – пригодность для занятий тем или иным видом спорта и ожидаемые при этом результаты. В то же время сообщения, несущие одну и ту же информацию, образуют целый класс эквивалентных сообщений. Например, прогноз погоды может быть получен по радио, из газеты, по телевидению, из Интернета и т. д.

Исходя из того, что информация фиксируется на материальных носителях в виде данных, и одни и те же данные могут нести различную информацию, следует важное положение о том, что данные могут обрабатываться при помощи различных технических средств. Причем такая обработка может осуществляться формальными методами, поскольку данные не зависят от содержащейся в них информации. Из всех технических средств обработки данных решающая роль, конечно же, принадлежит компьютерам. Данные в ЭВМ обрабатываются формально, без учета их смыслового содержания, а лишь с использованием математических и логических операций. Оценить смысловое содержание данных в настоящее время может только человек, который находится за пределами системы обработки данных. Те есть, обработка данных не всегда является обработкой содержания, а преобразование данных в информацию предполагает наличие соответствующего механизма интерпретации.

Соответствие между данными и содержащейся в них информацией называется ***правилом (методом) интерпретации***. Таким образом, одни и те же данные, по-разному интерпретированные, могут передавать разную информацию. Правило интерпретации может быть известно лишь ограниченному кругу лиц. Связь между данными и информацией особенно отчетлива в криптографии: никто посторонний не должен суметь извлечь из них информацию. Исходя из факта взаимодействия данных и методов

интерпретации, ей можно дать еще одно определение: ***информация*** – *это результат взаимодействия данных и адекватных им методов интерпретации*.

9. Передача информации

Информация не может существовать без наличия источника и приемника (получателя, потребителя) информации. *Источник информации* – это субъект или объект, порождающий информацию и представляющий ее в виде сообщения. *Приемник информации* – это субъект или объект, принимающий сообщение и способный правильно его интерпретировать.

В этих определениях сочетание «субъект» или «объект» означает, что источники и приемники информации могут быть одушевленными (человек, животные) или неодушевленными (технические устройства, природные явления). Для того чтобы объект (или субъект) считался источником информации, он должен не только ее породить, но и иметь возможность создать сообщение. Например, если человек что-то придумал, но держит это в своей голове, он не является источником информации. Однако он им становится, как только изложит свою идею на бумаге (в виде текста, рисунка, схемы и пр.) или выскажет вслух.

В определении приемника информации важным представляется то, что факт приема сообщения еще не означает получение информации. Информация может считаться полученной только в том случае, если приемнику известно правило интерпретации сообщения. Другими словами, понятия «приемник сообщения» и «приемник информации» не тождественны. Например, слыша речь на незнакомом языке, человек оказывается приемником сообщения, но не приемником информации.

Конечным источником и потребителем информации при ее обмене всегда является человек. Следовательно, воспринимать сообщения мы можем только посредством одного из пяти органов чувств, или некоторой их группой. Это не означает, однако, что человек не может использовать для приема и передачи информации какие-то иные сигналы, непосредственно им не воспринимаемые,

например радиоволны. В этом случае человек-источник использует промежуточное устройство, преобразующее его сообщение в радиоволны – радиопередатчик, а человек-приемник – другое промежуточное устройство – радиоприемник, преобразующий радиоволны в звук. Такой подход заметным образом расширяет возможности человека в осуществлении передачи и приема информации. Промежуточные устройства-преобразователи получили название *технические средства связи*, а в совокупности с соединяющей их средой они называются *каналом связи* (*каналом передачи информации, информационным каналом*). К ним относятся телеграф, телефон, радио, телевидение, компьютерные телекоммуникации и пр. Передача информации по каналам связи часто сопровождается воздействием помех, вызывающих искажение и потерю информации (рисунок 2.2.).

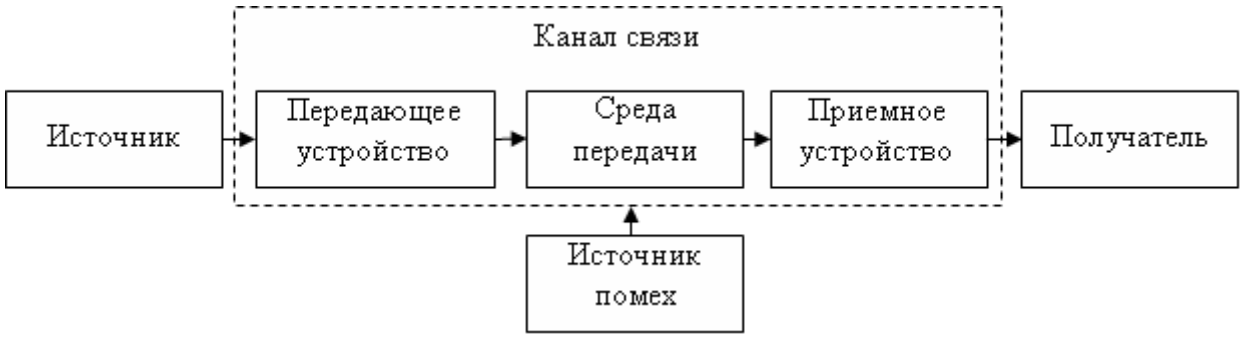


Рисунок 2.2 – Общая схема передачи информации

Примером канала связи может служить почта. Информация, записанная в виде текста на листе бумаги, помещается в конверт, опускается в почтовый ящик, затем извлекается из него и перевозится в почтовое отделение, где аналогичные отправления сортируются по направлениям назначения. Далее это отправление при помощи различных видов транспорта (автомобилей, поездов, самолетов и т.д.) доставляется в пункт назначения, где аналогичная корреспонденция распределяется по почтовым отделениям, откуда непосредственно доставляются адресату. Таким образом, почтовый канал связи включает в себя: конверт, транспорт, сортировочные машины, и почтовых работников. Информация, переданная при помощи этого канала связи, не искажается.

Другим примером может служить телефон. При этом источником сигнала является говорящий. Кодирующим устройством, преобразующим слова говорящего в электрические импульсы, является микрофон. Канал, по которому передается информация – телефонная сеть. Та часть телефона, которую мы подносим к уху, является декодирующим устройством, в котором электрические сигналы снова преобразуются в звук. Таким образом, информация поступает в «принимающее устройство» – ухо человека на другом конце канала. Канал связи включает в себя телефонные аппараты, соединительные провода, коммутирующие устройства АТС, приемо-передающие устройства и т.д. Особенностью этого информационного канала является то обстоятельство, что при поступлении в него информации в виде звуковых волн (речи), она преобразуется в электрические колебания, и лишь затем – передается. Такой канал называется *каналом с преобразованием информации*.

Еще один пример канала связи – компьютер. Его так же можно рассматривать как информационный канал с преобразование информации, поскольку, информация, поступающая с внешних устройств (клавиатура, микрофон и т.д.) преобразуется во внутренний код, обрабатывается, а затем снова преобразуется к виду, который может воспринимать устройство вывода (монитор, принтер и т.д.) и передается на них.

10. Формы представления информации

Формы представления информации в современном мире весьма многообразны. Информация может существовать в виде:

- текстов, рисунков и чертежей;
- световых и звуковых сигналов;
- электрических и нервных импульсов;
- жестов и мимики;
- запахов и вкусов;
- хромосом, передаваемых по наследству свойства организма и т.д.

При этом форма представления информации зависит как от источника, так и от приемника информации, а также канала связи между ними. Так, при общении между людьми обмен информацией происходит в форме, которую могут воспринимать наши органы чувств. Это могут быть речь, мимика, жесты, прикосновения и т.д. При обмене информацией между человеком и техническим устройство могут использоваться звук, различные зрительные образы и т.д. Обмен информацией между техническими устройствами осуществляется при помощи сигналов. При этом по информационным каналам связи могут передаваться два типа сигналов:

1. непрерывные (аналоговые) (рисунок 2.3а) и
2. дискретные (цифровые) (рисунок 2.3б).

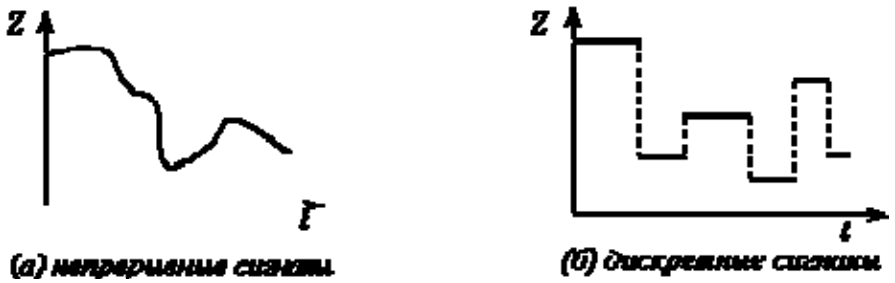


Рисунок 2.3 – Непрерывные и дискретные сигналы

Сигнал называется *непрерывным* (или *аналоговым*), если его параметр может принимать любое значение в пределах некоторого интервала. Примерами непрерывных сигналов являются речь, музыка, изображение, показание термометра (параметр сигнала – высота столба спирта или ртути – имеет непрерывный ряд значений) и пр. Примерно до 70-х годов XX века технические устройства преимущественно работали с аналоговыми сигналами. Это, например, запись и воспроизведение звука, его прием и передача на большие расстояния и т. д.

Дискретность сигнала означает, что он представляет собой не непрерывную функцию, а последовательность определенных значений. Дискретный сигнал называется *цифровым*, если его параметр может принимать конечное число значений в пределах некоторого пронумерованного диапазона. Сообщение, передаваемое с помощью таких сигналов, то же называется

дискретным, и информация, передаваемая источником в этом случае, также является дискретной. Пример дискретного сообщения – процесс чтения книги, информация в которой представлена текстом, т. е. дискретной последовательностью отдельных знаков (букв).

Принципиальным и важнейшим различием непрерывных и дискретных сигналов является то, что дискретные сигналы можно обозначить, т. е. приписать каждому из них некоторый *знак*, который будет отличать данный сигнал от других сигналов. *Знаком называется элемент некоторого конечного множества отличных друг от друга сущностей*. Природа знака может любой – жест, рисунок, буква, сигнал светофора, определенный звук и т. д. Его природа определяется носителем данных и формой представления информации в нем.

Вся совокупность знаков, используемых для представления дискретной информации, называется *набором знаков* – т. е., набор есть дискретное множество знаков. Набор знаков, в котором установлен порядок их следования, называется *алфавитом*. То есть, *алфавит – это упорядоченная совокупность знаков*. Порядок следования знаков в алфавите называется *лексикографическим*. Благодаря этому порядку между знаками алфавита можно установить отношения «больше–меньше». Для двух знаков «а» и «я» принимается, что ***а*** < ***я***, если порядковый номер у ***а*** в алфавите меньше, чем у ***я***. Примером алфавита так же может служить совокупность арабских цифр 0,1…9. С помощью такого алфавита можно записать любое целое число в системах счисления от двоичной до десятичной. Поскольку при передаче сообщения параметр сигнала должен меняться, очевидно, что минимальное количество различных его значений равно двум и, следовательно, *алфавит может содержать минимум два знака*. Такой алфавит называется *двоичным*.

 Представляется важным еще раз подчеркнуть, что *понятия знака и алфавита можно отнести только к дискретным сообщениям*.

Любое непрерывное сообщение может быть представлено непрерывной функцией, заданной на некотором интервале. Непрерывное сообщение можно преобразовать в дискретное. Эта процедура называется *дискретизацией* (или

оцифровкой). Из бесконечного множества значений параметра непрерывного сигнала выбирается их определенное число, которое приближенно может характеризовать остальные значения. Для этого диапазон изменения функции разбивается на отрезки равной длины и на каждом из этих отрезков значение функции принимается постоянным и равным (например, среднему значению на каждом отрезке). В итоге получается конечное множество чисел. Таким образом, любое непрерывное сообщение может быть представлено как дискретное, иначе говоря, последовательностью знаков некоторого алфавита.

Возможность дискретизации непрерывного сигнала с любой желаемой точностью (для возрастания точности достаточно уменьшить шаг) принципиально важна с точки зрения информатики. Компьютер – это цифровая машина, т.е. внутреннее представление информации в нем дискретно. Поэтому дискретизация входной непрерывной информации позволяет сделать ее пригодной для компьютерной обработки.

Существуют, однако, и другие вычислительные машины – аналоговые ЭВМ. Они используются обычно для решения задач специального характера и столь широкого распространения, как цифровые, не получили. Эти ЭВМ в принципе не нуждаются в дискретизации входной информации, так как ее внутреннее представление у них непрерывно. В этом случае все наоборот – если внешняя информация дискретна, то ее «перед употреблением» необходимо преобразовать в непрерывную.

11. Экономическая информация

11.1. Понятие экономической информации

Каждая область человеческой деятельности связана со «своей» информацией. Социально-экономическая и организационно-управленческая сферы деятельности оперирует информацией, которая называется *экономической*. Понятие «экономическая информация», с одной стороны, соответствует общему понятию «информация», а с другой стороны – неразрывно связана с экономикой и сферой материального производства.

Поэтому экономическая информация обладает свойствами, присущими как всей информации вообще, так и некоторыми характерными особенностями и отличиями, вытекающими из ее природы. Она так же является одной из разновидностей управленческой информации.

Под экономической информацией понимается информация, которая характеризует производственные отношения в обществе. К ней относятся различные полезные сведения, которые циркулируют в экономической сфере, и которые через систему натуральных, трудовых и стоимостных показателей, отображают процессы производства, распределения, обмена и потребления материальных благ, а также используются для управления этими процессами. Экономическая информация используется на всех уровнях управления народным хозяйством, в частности во всех сферах управления городом.

11.2. Классификация экономической информации

Экономическую информацию принято подразделять по следующим основным признакам:

- функциям управления,
- стадиям образования,
- стабильности во времени,
- объективности отображения явлений,
- полноте (насыщенности),
- поступлению и
- месту возникновения (уровню управления).

В зависимости от функций управления различают:

1) плановую,
2) учетную,
3) нормативно-справочную,
4) отчетно-статистическую и
5) регулирующую информацию.

Плановая информация содержит директивные указания о развитии конкретного объекта управления и его составляющих на конкретный период. В структуре экономической информации она занимает 8 – 10%.

Учетная информация в системе экономической информации охватывает в среднем 40 – 50%. Она отображает фактические значения плановых показателей на определенную дату. Составными частями учетной информации являются бухгалтерские и оперативные данные в виде натуральных, трудовых и стоимостных показателей.

Нормативно-справочная информация содержит различные нормативные и справочные данные, связанные с производственными процессами и отношениями. На ее долю приходится в среднем 30 – 40% от общего объема циркулирующей информации.

Отчетно-статистическая информация отражает результаты деятельности предприятия за отчетный период для вышестоящих органов управления, налоговой инспекции, органов государственной статистики и т.д. В общем объеме информации она занимает в среднем 5 – 10%.

Регулирующая информация занимает в среднем около 2% от общего объема экономической информации. На ее основе принимают решение относительно регуляции параметров производства или плановых заданий.

По стадиям образования экономическая информация бывает:

1) исходная (первичная) и
2) производная (вторичная).

Первичная информация отображает производственно-хозяйственные процессы в момент их прохождения. Это, как правило, бухгалтерская информация, сбор которой преимущественно осуществляется вручную и заносится на носитель типа «первичный документ».

Производная информация является результатом вычислений и, в свою очередь, делится на:

1) *промежуточную*, которая подлежит последующей обработке, и
2) *результативную*.

По стабильности во времени экономическая информация делится на:

1) переменную (рабочую, оперативную) и
2) постоянную (условно-постоянную).

К *переменной информации* принадлежат показатели разового использования, такие как, данные о количестве отработанного времени, выпущенной продукции и тому подобное. Показатели переменной (оперативной) информации имеют свойство изменять свои значения. Например, выработка одного и того же работника в разные дни, как правило, разная. Но норма выработки и расценка за труд могут быть одними и теми же. И тогда это уже постоянная информация. *Постоянная информация* используются многократно, то есть она характеризуются некоторой стабильностью. При этом важно отметить, что период стабильности носит вполне определенный характер для конкретных задач управления.

По объективности отображения явлений, событий, хозяйственных операций информация бывает:

1) достоверная и
2) недостоверная.

По полноте (насыщенности) она делится на:

1) недостаточную,
2) достаточную и
3) избыточную.

Информация может поступать работникам аппарата управления, как из собственных структурных подразделений, так и от других организаций. По этому признаку информацию делится на:

1) внутреннюю и
2) внешнюю.

И, на конец, по месту возникновения (уровню управления) экономическая информация подразделяется на:

1) входную и
2) выходную.

Входная информация – это информация, которая поступает в объект управления (предприятие, или его структурное подразделение) и используется как первичная (исходная) информация для реализации экономических и управленческих функций, а *выходная*, соответственно, – из объекта управления. При этом одна и та же информация может быть входной для одного объекта управления и одновременно выходной – для другого.

11.3. Свойства экономической информации

Экономическая информация обладает некоторыми особенностями, которые обусловлены ее природой. Принципиальное значение при создании информационных систем экономики имеют такие ее свойства:

- преобладание алфавитно-цифровых знаков;
- необходимость оформления результатов обработки данных в форме, удобной для восприятия человеком;
- широкое распространение бумажных документов как носителей входных и выходных данных;
- юридическое подтверждение подлинности информации соответствующими подписями или визами должностных лиц.

11.4. Структура экономической информации

Важной характеристикой экономической информации является ее структура. При этом выделяют:

1) простые и
2) составные

единицы информации.

Составной называют единицу информации, состоящую из совокупности других единиц информации, связанных между собой по смыслу.

Наименьшей, неделимой, *простой* единицей экономической информации является *реквизит*. Реквизитам присущи два основных свойства:

- отдельно взятый реквизит не может полностью характеризовать процесс или объект;

- отдельный реквизит может входить в состав различных экономических показателей.

Каждый реквизит характеризуется 3-я параметрами:

1. именем,
2. типом и
3. значением.

Имя реквизита служит для его однозначного выделения среди других реквизитов. Значение реквизита – это величина, которая характеризует некоторое свойство экономического объекта в конкретных обстоятельствах. Тип реквизита характеризует допустимое множество его значений, а также операции, которые над ним можно производить в процессе обработки.

В зависимости о характера отображаемой информации реквизиты делятся на:

- реквизиты-основания и
- реквизиты-признаки.

Реквизиты-основания отображают количественные характеристики явлений, процессов, хозяйственных операций. Они выражаются в цифровой форме. Над ними могут выполняться арифметические и логические операции. *Реквизиты-признаки* характеризуют качественные свойства экономических объектов, процессов и явлений. Они могут быть представлены в алфавитном, цифровом или алфавитно-цифровом виде. Над ними могут выполняться такие операции, как сортировка, поиск, группировка, выборка и т.д.

Реквизиты можно расчленить на более мелкие составляющие – например, символы или биты, но при этом теряется смысловое содержание реквизита, т.е. информация.

Логическая совокупность реквизитов-оснований и реквизитов-признаков составляет показатель. Показатель характеризует какой-либо объект, факт или процесс с количественной и качественной стороны. То есть, показатель, как основная составляющая экономической информации, включает один реквизит-основание (оценочная часть) и группу взаимосвязанных по смыслу, с ним и

между собой, реквизитов-признаков (призначная часть). Каждый показатель имеет наименование, которое раскрывает его основной экономический смысл. Обычно наименование указывает на экономический объект (прибыль, затраты и т. д.) и его основной признак (сумма, среднее и т. д.), например, прирост прибыли или сумма затрат. Кроме того, показатель содержит дополнительные признаки, которые уточняют его конкретное количественное значение. Состав дополнительных признаков определяется в каждом конкретном случае по-разному. Но обычно они выражаются в терминах единиц измерения (кг., шт., грн. и т. д.), времени, вида данных по функциям управления (плановый, фактический) и т. д.

Как минимальная смысловая единица экономической информации, показатели бывают:

1. *первичные* и
2. *вторичные*.

Первичные показатели отражают результаты производственно-хозяйственной деятельности объекта управления. Они определяются путем измерения, подсчета, взвешивания и т. д. К ним, например, относятся: количество отработанного времени, количество готовой продукции и т. д. Первичные показатели служат основой для формирования различного рода *вторичных показателей* – например, объем заработной платы, стоимость готовой продукции и т. д.

Поскольку показатель является основной смысловой составляющей экономической информации, то исключение хотя бы одной составляющей неизбежно влечет за собой потерю смысла. Вместе с тем в целях организации автоматизированной обработки экономической информации и реализации функций управления показатели могут образовывать более сложные структурные единицы, такие как, *документы*, *информационные массивы*, *информационные потоки*, *информационную базу*.

Документ – это организованная определенным образом совокупность взаимосвязанных экономических показателей. Он является основной и

наиболее удобной формой представления экономической информации с точки зрения управления, так как наряду с наглядностью представления информации, он содержит атрибуты, придающие ему юридический статус. Наиболее распространенной формой представления экономических документов является табличная форма, которая в самом общем виде включает:

1. общую (заголовочную),
2. предметную (содержательную) и
3. оформительскую

части (рисунок 2.4).

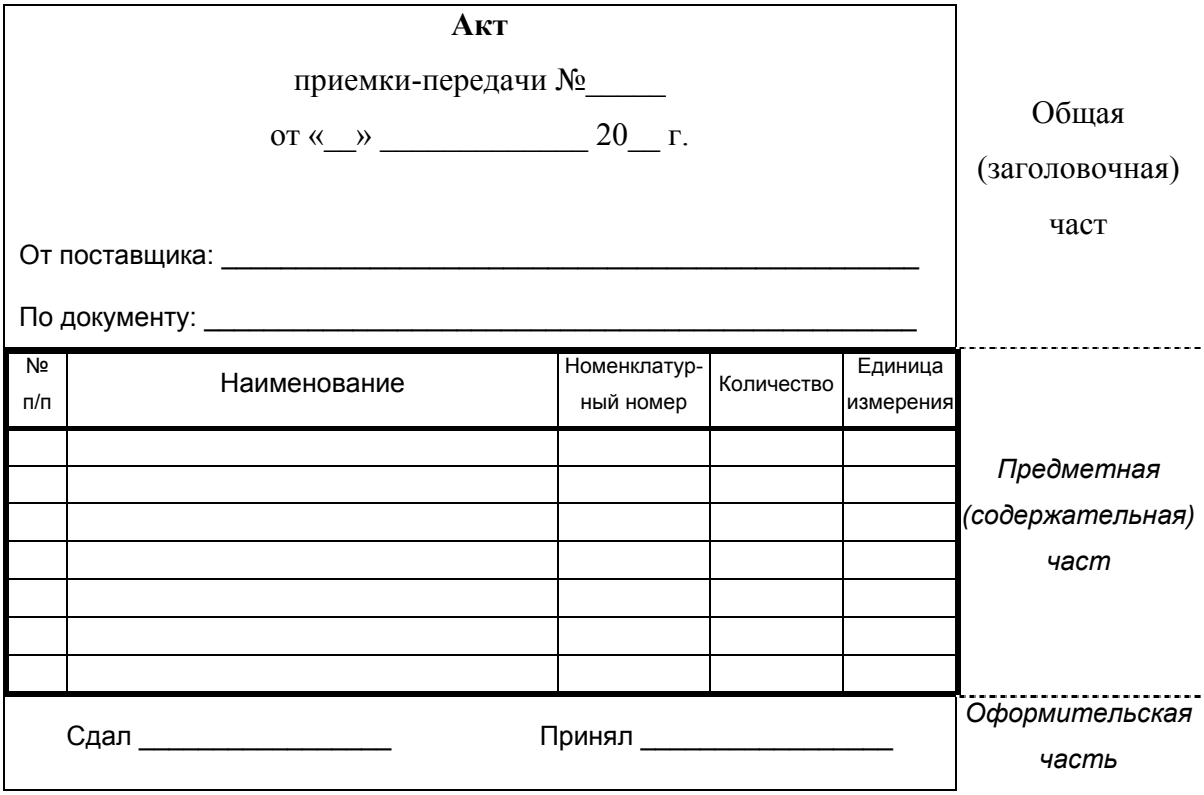


Рисунок 2.4 – Пример построения документа табличной формы

Общая (заголовочная) часть содержит перечень общих по составу и назначению реквизитов для всех показателей, представленных в документе. Как правило, это наименование предприятия, название документа, дата заполнения и т.д. Наличие общей части документа позволяет избежать дублирования показателей, входящих в состав многострочного документа.

Предметная (содержательная) часть включает реквизиты, характеризующие особенности экономических показателей многострочного документа. Она строится в виде таблицы, состоящей из строк и столбиков, в которых

размещаются переменные реквизиты-признаки и количественно-суммовые реквизиты-основания. Например, наименование, номенклатурный номер, количество, единица измерения и т. д.

Оформительская часть документа содержит атрибуты, непосредственно не участвующие в процессах обработки информации, однако они придают документу юридическую силу. Как правило, это подписи лиц, участвовавших в подготовке документа, а также подписи лиц согласующих и утверждающих данный документ.

Помимо табличной формы представления документов в практике организационно-экономического управления используются также документы упрощенной табличной формы. В них наименования реквизитов приводится не в шапке документа, а в боковике, рядом с которым проставляются конкретные значения соответствующих реквизитов.

Наконец, экономические документы могут содержать как шапку, так и боковик. Документы такой формы широко используются при подготовке различной отчетности (статистической, финансовой, бухгалтерской, налоговой и т. п.).

В качестве носителей (материальной основы) информации для отображения содержимого экономических документов наиболее распространенными являются:

1. *бумажные* и
2. *электронные*.

Следует отметить, что в последнее время особое внимание уделяется именно электронным способам отображения содержимого документов. Это позволяет значительно повысить эффективность систем управления благодаря использованию качественно новых подходов к реализации процессов обработки информации. При этом такие документы могут быть представлены как:

1. электронные копии,
2. электронные формы и

3. электронные документы.

Электронная копия является факсимильным отображением информации реальных бумажных документов. *Электронная форма* является компьютерной основой для решения задач управления, а в случае необходимости – основой для получения соответствующих бумажных аналогов, так называемых «твердых копий». В последнее время наиболее распространенной формой представления экономической информации является *электронный документ*. То есть – сведения, представленные в форме, воспринимаемой электронными средствами обработки информации и могущие быть преобразованными в форму, пригодную для восприятия человеком. Главный атрибут электронного документа (в отличие от электронной копии или электронной формы) – *электронная подпись*. Цель электронной подписи:

1. удостоверение подлинности сведений, отображенных на материальных носителях и
2. установления ее принадлежности к конкретному лицу.

Электронная подпись – это последовательность символов, имеющая неизменяемое соотношение с каждым символом определенного объема сведений электронного документа и предназначенная для подтверждения целостности и неизменности этого объема сведений, а также тождественности его содержания волеизъявлению заверившего его лица. Выработка электронной подписи осуществляется сертифицированными средствами электронной подписи, обеспечивающими такое ее формирование, при котором любое изменение в заверенном электронной подписью документе, нарушает его целостность и приводит к необходимости выработки новой электронной подписи.

В целях упрощения организации процессов обработки, передачи и хранения экономической информации, содержащейся в документах, она может объединяться в виде *информационных массивов*. Информационный массив, с позиции логической структуры, представляет собой набор данных (документов) одной формы (одного названия) со всеми их значениями, либо сочетание таких

наборов данных, относящихся к одной задаче. В первом случае – это однородный информационный массив, о втором случае – неоднородный. Сущность массива выражается через логический смысл и естественную целостность его структуры. В системах обработки информации массив является основной структурной единицей, предназначенной для хранения, передачи и обработки информации. Массивы могут объединяться в более крупные структурные единицы: информационные потоки и информационные базы. *Информационный поток* – это совокупность информационных массивов, в том числе документов, относительно конкретной управленческой деятельности, имеющая динамический характер. *Информационная база* – это вся совокупность информации реального экономического объекта.

Рассмотренные структурные единицы экономической информации отражают их *логическое построение* без учета особенностей представления данных на физических (материальных) носителях. При организации автоматизированной обработки экономической информации понятие структуры данных связывается с представлением их на различных носителях и таким образом соответствующие структурные единицы выделяются в зависимости от особенностей того или иного носителя и способов фиксации данных на нем. Это составляет основу *физического подхода* к рассмотрению структур информации.

ЛЕКЦИЯ № 3

СИСТЕМНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ ПРОЦЕССОВ

План

1. Состав и назначение программного обеспечения компьютера
2. Структура компьютерной программы
3. Операционная система Linux

3.1. История

3.2. Современный Linux

3.3. Ядро

3.4. Файловая система

3.5. Интерфейс

3.6. Утилиты

3.7. Дистрибутивы

4. Полиморфизм

1. Состав и назначение программного обеспечения компьютера

Секрет столь широкого использования компьютеров в различных областях человеческой деятельности кроется в программном обеспечении, которое позволяет использовать одну и ту же машину, построенную из кремния и стали, для решения огромного множества самых разнообразных задач. В отличие от первых программ, вводившихся в машину посредством утомительного переключения множества тумблеров и цифро-наборных устройств, программное обеспечение, вдохнувшее жизнь в современный компьютер, записывается, как правило, на магнитных дисках и вступает в работу сразу после включения машины. Однако по существу команды компьютера, хотя они и задаются теперь гораздо более удобным способом, заметных изменений не претерпели. Любой компьютер должен разложить задание на последовательность машинных команд, а затем выполнять их одну за другой.

В английском языке для программного обеспечения (ПО) выбрано (а точнее, создано) довольно удачное слово *SoftWare* (буквально – «мягкое изделие»), которое подчеркивает равнозначность программного и аппаратного обеспечения («железа» – *HardWare* – «жесткое изделие»), и вместе с тем говорит о его гибкости, способности модифицироваться, приспосабливаться, развиваться. Введение этих терминов было связанно с необходимостью провести четкую грань между командами-инструкциями, управляющими компьютером, и его физическими компонентами или аппаратным обеспечением, которое, собственно, и составляет компьютер. Поэтому переключение компьютера с построения рисунка на разработку стандартного контракта или с разработки архитектурного проекта на создание карты погоды земного шара осуществляется простым изменением последовательности команд, управляющих его работой, т. е. программ.

Прежде, чем обсуждать состава и назначения программного обеспечения, введем еще два термина: *программист* и *пользователь*. Если программа сложная, в ее создание вовлекается большая группа людей, выполняющих различные функции, то мы будем употреблять общий термин «программист» для обозначения человека, который проектирует, записывает и совершенствует новую программу, или модифицирует старую. Человек, для которого пишется программа и который, вероятно, будет снабжать ее входными данными и использовать полученные результаты, называется пользователем. И хотя один и тот же человек может быть и пользователем, и программистом, важно различать эти две ипостаси.

Общий термин программное обеспечение (ПО, *SoftWare*) применяется для обозначения программ, используемых в компьютерных системах. Все ПО можно классифицировать по множеству самых разнообразных признаков. Наиболее существенными из них являются классификации по:

- *назначению*, или месту использования в технологическом процессе обработки информации, и
- *виду* (*типу*) *лицензии*.

В зависимости от назначения программы подразделяются на два больших класса, которые называются, соответственно:

1. прикладное программное обеспечение (ППО) и
2. базовое программное обеспечение (БПО).

Прикладное программное обеспечение образуют программы для пользователей и осуществляющие информационные процессы, которые нужны пользователям. Оно предназначено для решения целевых задач пользователя. Так, например, в бизнесе наиболее распространенной прикладной программой является 1С:Предприятие, которая предназначена для решения различных экономических задач.

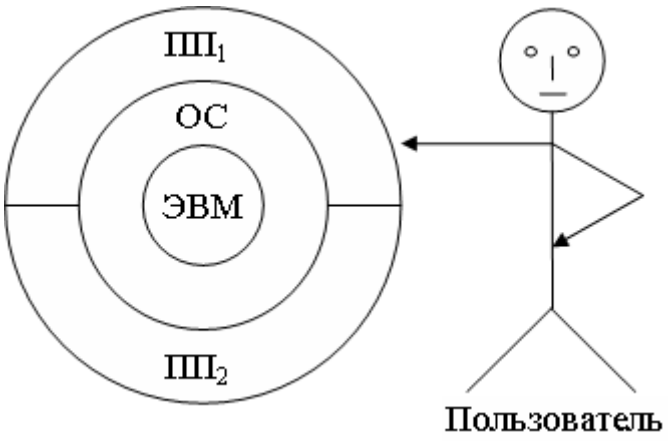
Базовое программное обеспечение – это «накладные расходы» компьютерной обработки информации. То есть, оно непосредственно не участвует в решении целевых задач пользователя. Но и без него эти проблемы решить не возможно.

Центральное место в базовом программном обеспечении занимает специальная совокупность программ, называемая *операционной системой*, которая обеспечивает взаимодействие:

1. человека,
2. компьютера и
3. программ.

Такое взаимодействие выполняется в соответствии с рисунком 3.1.

Операционная система дает указание компьютеру, как интерпретировать команды и данные, как распределять аппаратные ресурсы для выполнения заданий и как управлять периферийными устройствами (например, принтером или дисплеем). Она также обеспечивает возможность непосредственного взаимодействия человека и компьютера, выполняя такие действия, как хранение программ и данных на внешних запоминающих устройствах.



ЭВМ – электронная вычислительная машина (компьютер)
ОС – операционная система
$ПП_1$, **$ПП_2$** – прикладная программа1, прикладная программа2

Рисунок 3.1 – Взаимодействие пользователя, компьютера и программ

Если рассматривать операционную систему как «режиссера» компьютерного действа, то прикладные программы играют роль «артистов». Именно благодаря таким программам, как текстовые процессоры, игры и электронные таблицы, компьютер приобретает удивительную разносторонность.

Понятие «базовое программное обеспечение» включает также такие программы, как трансляторы и утилиты. Транслятор – это программа, которая принимают в качестве исходных данных другую программу, написанную на так называемом языке программирования высокого уровня, отдаленно напоминающем язык человека, и отличном от машинного кода, с которым имеет дело процессор, и переводит эту программу в машинный язык, представляющий собой комбинацию нулей и единиц, которые компьютер обрабатывает как последовательности электрических импульсов. Утилиты выполняют рутинные, но часто крайне необходимые, функции, например, удаление ненужной информации с магнитных дисков. Эти скромные «рабочие лошадки» помогают решать типовые задачи обработки информации.

И так, полная *вычислительная система*, включающая аппаратуру (*HardWare*) и программное обеспечение (S*oftWare*), создает среду, в которой могут разрабатываться, храниться и выполняться программы. Границы между отдельными слоями в этой вычислительной системе (рисунок 3.1) называются

интерфейсами. То есть, *интерфейс – это набор правил взаимодействия между отдельными компонентами системы*. Так, например:

- граница между пользователем и прикладной программой называется *Графическим Интерфейсом Пользователя* (англ. *Graphical User Interface – GUI*),
- граница между прикладной программой и операционной системой – *Интерфейсом Прикладного Программирования* (англ. *Application Programs Interface – API*), а
- границу между операционной системой и компьютером образует так называемая *система (набор, архитектура) команд центрального процессора*.

При этом, одни операционные системы могут работать только с одной, вполне определенной, системой команд, а другие – поддерживать различные системы команд. Например, операционные системы класса Microsoft Windows «понимают» только так называемую систему команд *x86* (процессоры корпораций Intel и AMD, на основе которых создаются персональные компьютеры IBM PC), в то время как операционные системы семейства Linux могут работать с различными системами команд.

В ситуации, когда программное обеспечение является объектом продажи наравне с предметами обихода, на него автоматически распространяются уже не только законы научной разработки, но и свойства материальных предметов, которыми можно торговать, обмениваться, право владения и пользования которыми охраняется законном. Так программное обеспечение попало в разряд *интеллектуальной собственности*, т. е. программы стали рассматриваться как объекты применения *авторского права*.

Чтобы защитить свои интересы, производители программного обеспечения используют лицензии – особый вид договора между обладателем авторских прав и пользователем (покупателем) программного обеспечения. Лицензия на программное обеспечение – это правовой инструмент, определяющий порядок распространения и использования программного

обеспечения. По сути, лицензия выступает гарантией того, что издатель ПО, которому принадлежат исключительные права на программу, не подаст в суд на того, кто ею пользуется.

Лицензии на программное обеспечение, в целом, делятся на две большие категории:

- *несвободные* (они же *собственнические*, *проприетарные*, *полусвободные*) и
- *лицензии свободного и открытого ПО*.

Их различия сильно влияют на права конечного пользователя в отношении использования ПО.

Основной чертой проприетарных лицензий является то, что издатель ПО в лицензии дает разрешение ее получателю использовать одну или несколько копий программы, но при этом сам остается правообладателем всех этих копий. Одно из следствий такого подхода заключается в том, что практически все права на ПО остаются за издателем, а пользователю передается лишь очень ограниченный набор строго очерченных прав. Типичной проприетарной лицензией может служить лицензия на операционную систему Microsoft Windows, которая включает большой список запрещенных вариантов использования, таких, например, как установка одной ее копии на нескольких компьютерах.

В отличие от проприетарных, свободные и открытые лицензии не оставляют права на конкретную копию программы ее издателю, а передают самые важные из них конечному пользователю, который и становится ее владельцем. Примером наиболее распространенной свободной лицензии является GPL (General Public License), которая дает пользователю право самому распространять ПО под этой лицензией, участвовать в его разработке или изменять другими способами. Тем не менее, перечисленные права обязывают пользователя ПО под GPL подчиняться определенным правилам. Например, любые изменения программы, сделанные пользователем и распространенные дальше, должны сопровождаться исходным кодом этих изменений.

2. Структура компьютерной программы

Любая компьютерная программа состоит из 3-х частей (рисунок 3.2):

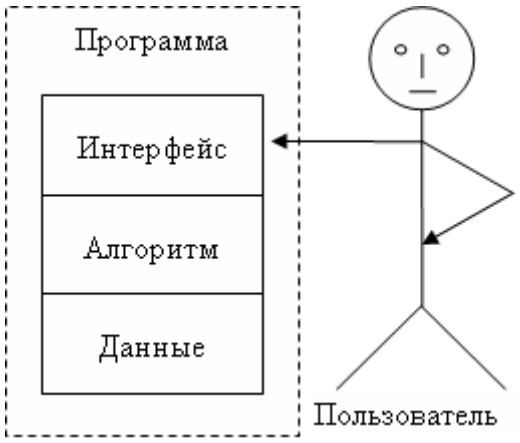


Рисунок 3.2 – Структура компьютерной программы

1. *Интерфейс* – набор правил взаимодействия между пользователем и программой;
2. *Алгоритм*, или *бизнес-логика* – набор правил преобразования информации из входной в выходную;
3. *Данные* – то, что обрабатывает программа.

Все три части обязательно присутствуют в любой компьютерной программе. Однако, «удельный вес» каждой из них зависит от назначения программы. Так, например, в научно-технических (инженерных) программах преобладают алгоритмы, а интерфейс и данные у них, как правило, несложные. Экономические задачи, наоборот, обрабатывают большие объемы информации (т. е. преобладают данные), а алгоритмы и интерфейс у таких программ относительно простые. Игровые программы обладают хорошо проработанными интерфейсами и захватывающими алгоритмами, и несложными данными.

3. Операционная система Linux

Linux является одной из выдающихся представительниц древнего и славного рода операционных систем UNIX. Свое начало род ведет от 1969 года, когда сотрудники Bell Labs (одного из исследовательских подразделений американской компании AT&T) Кен Томпсон и Деннис Ритчи создали самый

первый вариант операционной системы, которая в дальнейшем получила название UNIX.

3.1. Краткая история

К началу 80-х годов XX-го века операционная система UNIX стала наиболее распространенной ОС в мире, особенно в академической среде и Интернете. Для этой операционной системы существовало множество программ, свободно распространявшихся в научном сообществе. Но, особенность данной ситуации состояла в том, что ОС UNIX в то время была несвободным программным продуктом.

Начало решения этой проблемы было положено Ричардом Столлманом, который 27 сентября 1983 года в группах новостей net.unix-wizards и net.usoft опубликовал объявление о начале разработки операционной системы *GNU*. Целью проекта GNU было создания «целостной UNIX-совместимой программной системы», полностью состоящей из свободного программного обеспечения. Как писал позже сам Столлман: «Аббревиатура GNU расшифровывается как «GNU – это не UNIX» (GNU's – Not UNIX). Основная идея, заложенная в процесс разработки системы GNU – это ее полное отделение от UNIX. UNIX всегда была и остается несвободным ПО…». Т. е. то, что принадлежит проекту GNU, не является частью UNIX (потому что к тому времени даже само слово UNIX уже было зарегистрированной товарной маркой, т. е. перестало быть свободным).

В рамках проекта GNU было создано большинство компонент, необходимых для функционирования свободной операционной системы, и свободно распространяемых в Интернете. Помимо текстового редактора *Emacs*, Столлман создал компилятор *gcc* (GNU C Compiler) и отладчик *gdb*. Будучи выдающимся программистом, Ричард Столлман в одиночку сумел создать эффективный и надежный компилятор, который превосходит по своим качествам продукты коммерческих поставщиков, создаваемые целыми коллективами программистов. Как пишет Р. Столлман «К 1990 году система GNU была практически закончена; не хватало только одного из базовых

компонентов – ядра». Т. е. для превращения GNU в полноценную операционную систему не хватало только одного ядра.

Разработка ядра велась (оно называлось Hurd), но по каким-то причинам задерживалась. Поэтому появление ядра, разработанного финским студентом Линусом Торвальдсом (Linus Torvalds) было очень своевременным. Оно «упало на подготовленную почву» и ознаменовало рождение новой операционной системы с полностью открытым исходным кодом, которая в последствии получила название Linux. Линус Торвальдс оказался со своей разработкой в нужном месте в нужное время.

Непосредственным же прообразом (прототипом) для Linux послужила операционная система MINIX – один из вариантов UNIX-подобной операционной системы. MINIX была разработана в 1987 году Эндрю Таненбаумом (Andrew S. Tanenbaum), профессором Университета Врие (Vrije University), Амстердам, Нидерланды. Свою операционную систему Таненбаумом разработал как учебное пособие, на примере которого он показывал студентам внутреннее устройство реальной операционной системы.

Конечно, как операционная система MINIX не была верхом совершенства. Но у нее было одно очень важное качество – ее исходные коды были открыты. MINIX можно было приобрести вместе с книгами Таненбаума или отдельно, и она могла быть реально установлена на персональный компьютер. Великолепный автор, Таненбаум сумел вовлечь самые выдающиеся умы компьютерной науки в обсуждение искусства создания операционных систем. Студенты компьютерных факультетов по всему миру корпели над книгами Таненбаума, вчитываясь в коды с целью понять, как работает та самая система, которая управляет их компьютером. И одним из таких студентов был Линус Торвальдс.

Официальным днем рождения операционной системы Linux считается 25 августа 1991 года, когда Линус Торвальдс направил первое сообщение о своей разработке в группу новостей посвященную ОС MINIX comp.os.minix. А 17 сентября 1991 года он выложил в Интернете исходный код программы

(версии 0.01) для общедоступной загрузки. Каталог на FTP-сервере, где была выложена система, назывался pub/OS/Linux. И это название постепенно закрепилось за новой операционной системой.

Если же быть совершенно точным, то слово «Linux» означает только ядро. Поэтому, когда речь идет об операционной системе, правильнее было бы говорить об «операционной системе, основанной на ядре Linux». Ричард Столлман, основатель движения свободного ПО, настаивает на том, что операционную систему следует называть не Linux, а GNU/Linux. Но название Linux исторически уже закрепилось за этой ОС, поэтому далее тоже будем называть ее просто Linux (не забывая о заслугах Р. Столлмана и его сподвижников).

3.2. Современный Linux

Современный Linux – это многопользовательская, многозадачная операционная система. Это означает, что одновременно на одном компьютере может работать много пользователей, каждый из которых одновременно может запускать на выполнение много различных программ. На современном персональном компьютере одновременно может работать только один человек. Однако многопользовательская модель имеет много преимуществ. Например, один пользователь может запустить длительный процесс (скачивание фильма из Интернета) и предать компьютер другому, или один и тот же пользователь может зарегистрироваться в системе под разными именами, и с разными полномочиями, для выполнения различных задач и т. д.

В любой операционной системе общего назначения, и Linux в том числе, можно выделить 4-е основные составляющие:

- *ядро*,
- *файловую систему*,
- *интерфейс* и
- *утилиты*.

3.3. Ядро

Ядро (англ. kernel) – это основная, определяющая, часть ОС, которая управляет работой аппаратной части компьютера и выполнением программ. Она распределяет ресурсы компьютера между одновременно выполняющимися программами, а также обеспечивает их бесконфликтное выполнение. Обычный пользователь с ядром непосредственно не работает. Он взаимодействует с такими компонентами ОС как интерфейс, файловая система и утилиты. В основном с ядром работают программисты и системные администраторы. Ядро ОС Linux, в отличие от остальных составляющих системы, в течение всего времени разрабатывается и совершенствуется централизовано под общим руководством Линуса Торвальдса. В настоящее время лишь около двух процентов ядра написано самим Торвальдсом, но за ним по-прежнему остается решение о внесении изменений в официальную ветку.

3.4. Файловая система

Файловая система представляет собой способ (метод) хранения и организации доступа к данным на внешних запоминающих устройствах (ВЗУ) компьютера (таких, например, как магнитные и оптические диски, флеш-накопители и т. д.). Любая файловая система состоит из:

1. набора правил хранения информации на ВЗУ и
2. набора программ, которые эти правила реализуют.

Каждый такой набор правил имеет свое уникальное имя – *название файловой системы*. «Родными» для операционной системы Linux являются файловые системы *Ext3fs/Ext4fs* (англ. Extended File System ver. 3/4 – «расширенная файловая система версий 3 и 4»). Однако Linux поддерживает работу и иных файловых систем, таких, например, как FAT, NTFS, CDFS и др., что обеспечивает совместимость (прямой обмен информацией) с другими операционными системами. Программы, которые реализуют работу файловых систем, входят в состав так называемой подсистемы ввода-вывода операционной системы.

Основными объектами файловой системы Linux являются:

1. *файлы* и
2. *каталоги*.

 *В файловой системе Linux нет ни дисков, ни папок, ни документов – есть только каталоги и файлы.*

Файл – это поименованная совокупность данных на ВЗУ. В других операционных системах файлы еще иногда называют документами. Каждый файл имеет множество различных характеристик. Основной среди них, с точки зрения пользователя, являются имя файла. Имя файла, обычно, каким-то образом отражают его содержание. В Linux, в отличие от других операционных систем, не существует какого-либо формата относительно имен файлов (как, например, «8.3» в MS-DOS). В ней нет никаких предписаний по поводу расширения имени файла: в имени файла может быть любое количество точек (в том числе и ни одной), а после последней точки может быть любое количество символов. Наоборот, если имя файла начинается с символа точка «.» – он становится «невидимым» для некоторых команд. Имя файла в Linux может иметь длину до 255 символов и содержать любые символы, за исключением двух символов: с кодом 0 и наклонной черты (слеш) «/». Причем,

 *Linux всегда различает прописные и строчные буквы, как в именах файлов, так и в именах каталогов*.

Есть несколько символов, допустимых в именах файлов и каталогов, которые, при этом, нужно использовать с осторожностью. Это – так называемые *спецсимволы*: «~», «`», «!», «@», «#», «$», «%», «&», «*», «(», «)», «[», «]», «{», «}», «\», «|», «;», «:», «'», «"», «<», «>», а также символы пробела и табуляции. Если же все-таки используется один или несколько таких символов, то все имя необходимо заключить в двойные кавычки «"».

Однако с точки зрения ОС Linux имя файла отнюдь не является его основной характеристикой. В файловой системе Linux каждому файлу однозначно соответствует так называемый «*индексный дескриптор*» (англ. *inode*). Компьютеру удобнее работать с числами, а человек лучше воспринимает символические имена. Из этого следует, что в Linux каждый

файл может иметь несколько *символических имен* (*ссылок*). При этом ссылки бывают двух типов:

1. *жесткие* (англ. *hard links*) и
2. *символические* (англ. *symbolic links*).

Жесткие ссылки – это, просто, различные имена одного и того же содержания. Количество жестких ссылок на одну и ту же область данных (файл) не ограничено, т. е. у файла может быть много разных имен. Первая жесткая ссылка (первое имя) формируется при создании файла. Следующие жесткие ссылки создаются с помощью команды «`ln`» (от англ. «link» – «соединять, связывать»), которая имеет два параметра: первый параметр – это имя файла, на который нужно создать ссылку, а второй – имя новой ссылки. После создания жесткой ссылки невозможно различить, где исходное имя, а где ссылка. Когда удаляется файл, имеющий несколько разных имен – жестких ссылок, то фактически удаляется только одна ссылка – та, которая указана в команде удаления файла. Для того, чтобы удалить файл полностью (т. е. освободить занимаемую им область, и не иметь больше возможности обращаться к этим данным) необходимо удалить все его имена – все жесткие ссылки. Номер индексного дескриптора любого файла можно узнать при помощи команды «`ls`» с ключом «`-i`».

⚠ *Жесткие ссылки создаются только в пределах одного носителя (одной файловой системы), их также нельзя создавать на каталоги*.

Символьная (символическая) ссылка – это просто файл, в котором содержится имя другого файла. Они называются так потому, что содержат символы – путь к файлу или каталогу. Символьные ссылки, как и жесткие, предоставляют возможность обращаться к одному и тому же файлу по разным именам. Кроме того, *символьные ссылки могут указывать на файлы и каталоги в других файловых системах (т. е. на других носителях, или даже на других компьютерах), чего не позволяют жесткие ссылки*. Символьные ссылки в Linux аналогичны ярлыкам в Microsoft Windows. Если исходный файл или каталог удалены, символьная ссылка не удаляется, но становится бесполезной –

она не будет «работать». Например, если попробовать вывести содержимое такой «битой ссылки» при помощи команд «`cat`» или «`ls`», будет выдано сообщение об ошибке. Создаются символьные ссылки при помощи той же команды, что и жесткие – «`ln`», только с ключом «`-s`» (от англ. «symbolic» – «символический»).

До тех пор, пока в системе количество файлов не слишком большое, не существует и проблемы их поиска. Если же количество файлов возрастает, например, до нескольких тысяч, то остро встает вопрос их упорядочивания и систематизации. Для решения этой задачи и были созданы (придуманы) *каталоги*.

Понятие *каталога* (англ. *directory*) позволяет систематизировать все объекты, размещенные на носителе данных (например, на диске). В большинстве современных файловых систем используется древовидная структура организации каталогов, т. е. каталоги могут содержать другие каталоги («У больших каталогов есть маленькие каталоги…» Огастес Де Морган). Каталог, на который есть ссылка в данном каталоге, называется подкаталогом, вложенным или дочерним каталогом. Каталог, который содержит текущий каталог, называется родительским каталогом.

В любой файловой системе Linux (и UNUX вообще) всегда есть только один корневой каталог («корень», «root»), который обозначается символом «`/`». Пользователь Linux всегда работает с единым деревом каталогов, даже если разные данные расположены на разных носителях: нескольких жестких или сетевых дисках, съемных дисках, CD-ROM и т. п. *Для того, чтобы подключать и отключать файловые системы на разных устройствах в одно общее дерево, используются процедуры монтирования и размонтирования (которые может выполнять только пользователь с правами администратора – «`root`»).* После того, как файловые системы на разных носителях подключены к общему дереву, содержащиеся на них данные доступны так, как если бы все они составляли единую файловую систему: пользователь может даже не знать, на каком устройстве какие файлы хранятся.

Организация каталогов файловой системы в виде дерева не допускает появления циклов: т. е. каталог не может содержать в себе каталог, в котором содержится сам. Благодаря этому ограничению путь до любого каталога или файла всегда будет однозначным и конечным.

Имена каталогов в Linux строятся по тем же правилам, что и имена файлов. И, вообще, каталоги в принципе ничем, кроме своей внутренней структуры, не отличаются от «обычных» файлов, например, текстовых. Т. е. каждый каталог в Linux – это отдельный файл особого типа («`d`» от англ. «directory»), отличающийся от обычного файла с данными тем, что в нем могут содержаться только ссылки на другие файлы и каталоги.

В каждый момент времени пользователь работает только с одним, так называемым, «*текущим*» каталогом. Текущий каталог – это каталог, в котором запущенные программы пользователя по умолчанию (т. е. если явно не указан другой каталог) читают и записывают данные. В Linux имеется специальная команда «`pwd`» (от англ. «print working directory» – «напечатать (вывести) рабочий каталог»), которая возвращает имя (полный путь) текущего каталога.

Путем (англ. «path, pathname»; «тропой» к файлу) называется список каталогов, который необходимо «пройти в дереве каталогов» (маршрут, траектория), чтобы попасть в целевой каталог. Каталоги в пути друг от друга отделяются тем же символом «`/`», который служит для обозначения корневого каталога. Поэтому символом «`/`» нельзя использовать в менах файлов и каталогов. Различают полный путь, который начинается в корневом каталоге (и начинается символом «`/`»), и относительный путь, который начинается в текущем каталоге (и, соответственно, не начинается символом «`/`»).

Кроме текущего каталога в ОС Linux для каждого пользователя определен еще его «*домашний каталог*» (аналог папки «*Мои документы*» Microsoft Windows) – каталог, предназначенный для хранения собственных данных пользователя. В этом каталоге пользователь имеет все права: может создавать и удалять файлы и каталоги, менять права доступа к ним и т. д. В каталоговой структуре Linux домашние каталоги пользователей обычно размещаются в

каталоге /home и имеют имена, совпадающие с именами пользователей. Например, /home/student21 – полный путь в домашний каталог пользователя student21. Каждый пользователь может обращаться к своему домашнему каталогу с помощью символа «тильда» – «~» (в системе не существует каталога с именем «~» – это просто «синтаксический сахар»). Поэтому строка «~/temp» означает каталог temp, который находится в домашнем каталоге. Когда пользователь входит в систему, текущим каталогом становится его домашний каталог.

На структуру каталогов UNIX-подобных операционных систем существует довольно строгий стандарт – так называемый Filesystem Hierarchy Standard (FHS). Он регламентирует не только расположение основных каталогов, но и их подкаталоги, а иногда даже приводит список конкретных файлов, которые должны присутствовать в определенных каталогах. Этот стандарт последовательно соблюдается во всех Linux-системах. Корневой каталог большинства ОС Linux имеет, примерно, такое содержание:

- /bin (от англ. «binaries» – «двоичные, исполняемые») – в этом каталоге находятся исполняемые файлы самых необходимых утилит (команд) Linux, таких, например, как «pwd», «ls», «mv» и т. п.
- /boot («загрузка системы») – в этом каталоге находятся файлы, необходимые для загрузки системы, в частности само ядро Linux.
- /dev (от англ. «devices» – «устройства») – в этом каталоге находятся все имеющиеся в системе драйверы устройств – они используются для доступа к устройствам и ресурсам системы, таким, например, как диски, модемы, память и т. д.
- /etc (от англ. «et cetera» – «разное, и так далее») – содержит различные служебные файлы, в частности, – системные конфигурационные файлы.
- /home («домой») – здесь расположены домашние каталоги пользователей.

- /lib (от англ. «libraries» – «библиотеки») – в этом каталоге находятся образы разделяемых библиотек (англ. shared library images) – файлы, содержащие код, который могут использовать многие программы.
- /lost+found («потерять и найти») – этот каталог используется при восстановлении файлов системы командой fsck.
- /mnt (от англ. «mount» – «монтировать») – каталог для монтирования – временного подключения файловых систем, например, на съемных носителях (CD-ROM и др.).
- /proc – это «виртуальная файловая система», в которой файлы хранятся в памяти, а не на диске; они связаны с различными процессами, происходящими в системе, и позволяют получить информацию о том, что делают программы и процессы в указанное время.
- /root – домашний каталог администратора системы – пользователя «root».
- /sbin (от англ. «system binaries») – в дополнение к утилитам /bin здесь находятся программы, необходимые для загрузки, резервного копирования и восстановления системы, полномочия, на исполнение которых, есть только у системного администратора.
- /tmp (от англ. «temporaries» – «временные файлы») – этот каталог предназначен для временных файлов: в таких файлах программы хранят промежуточные данные, необходимые им только на время работы; после завершения работы программы временные файлы теряют смысл и, как правило, удаляются.
- /usr – это «государство в государстве». Здесь можно найти такие же подкаталоги bin, etc, lib, sbin, как и в корневом каталоге. Однако в корневой каталог попадают только утилиты, необходимые для загрузки и восстановления системы в аварийной ситуации, все остальные программы и данные располагаются в подкаталогах /usr.

- `/var` (от англ. «variable» – «переменные») – здесь размещаются те данные, которые создаются в процессе работы разными программами и предназначены для передачи другим программам и системам (очереди печати, электронной почты и др.) или для сведения системного администратора (системные журналы, содержащие протоколы работы системы). В отличие от каталога `/tmp` сюда попадают те данные, которые могут понадобиться после того, как создавшая их программа завершила работу.

 Более подробно с основными каталогами Linux можно познакомиться по тексту стандарта FHS, полностью который можно найти по адресу http://www.pathname.com/fhs/.

Файловая система ОС Linux поддерживает следующие типы файлов, которые обозначаются первым символом в поле прав доступа, и принимающего такие значения:

- «`-`» – обычный файл – может содержать текстовые данные, музыку, фильм, программу и т. п.
- «`d`» – каталог – файл, который может содержать только список ссылок на другие вложенные файлы и каталоги.
- «`b`» – файл блочного устройства – устройство, запись и чтение информации в котором выполняется блоками, например, жесткий диск.
- «`c`» – файл символьного устройства – устройство, запись и чтение информации в котором выполняется посимвольно, например, терминал.
- «`s`» – доменное гнездо (от англ. «socket») – это соединение между процессами, которое позволяет им взаимодействовать, не подвергаясь влиянию других процессов. Они являются ключевым понятием TCP/IP и, соответственно, на них целиком строится Интернет. Среди стандартных средств, использующих гнезда – система *X Window*, система печати и система *syslog*.

- «p» – именованный канал (от англ. «pipe» – «труба») – или буфер FIFO (First In – First Out) служит в основном для того, чтобы организовать обмен данными между разными приложениями (процессами). Все, что один процесс помещает в канал, другой может оттуда прочитать. Именованные каналы создаются при помощи команды «mkfifo».
- «l» – символическая ссылка (от англ. «link») – файл, который может содержать только символический указатель (ссылку) на другой файл.

Поскольку Linux – система многопользовательская и многозадачная, вопрос разграничения прав доступа к файлам и каталогам является одним из существенных моментов функционирования операционной системы (необходимо защитить файлы каждого пользователя от «дурного влияния» других пользователей).

В основе механизма разграничения доступа лежат имена пользователей и имена групп пользователей. В Linux каждый пользователь имеет уникальное имя, под которым он входит в систему. Кроме того, в системе создается некоторое число групп пользователей, причем каждый пользователь может быть включен в одну или несколько групп. Члены разных групп могут иметь разные права по доступу к файлам и каталогам. Например, группа администраторов имеет больше прав, чем группа программистов. Создает и удаляет группы пользователей суперпользователь («root»), он же может изменять состав той или иной группы.

Права доступа подразделяются на три типа:

- чтение (англ. ***r**ead*),
- запись (англ. ***w**rite*) и
- выполнение (англ. *e**x**ecute*).

Эти типы прав доступа могут быть предоставлены трем классам пользователей:

- владельцу файла,
- группе, в которую входит владелец, и
- всем прочим пользователям.

Разрешение на чтение позволяет пользователю читать содержимое файла, а в случае каталога – просматривать перечень имен файлов в каталоге. Разрешение на запись позволяет пользователю писать в файл и изменять его. Для каталогов это дает право создавать и удалять в нем вложенные файлы и каталоги. Наконец, разрешение на выполнение позволяет пользователю выполнять файлы (как бинарные программы, так и командные файлы). Разрешение на выполнение применительно к каталогам означает: во-первых, сделать этот каталог текущим (команда «`cd`»), а во-вторых, обращаться за доступом к содержащимся в нем файлам.

Важно заметить, что права доступа, которые имеет файл, зависят также от прав доступа к каталогу, в котором этот файл находится. Например, даже если файл имеет права доступа «`rwxrwxrwx`», другие пользователи не смогут до него «добраться», если у них не будет прав на чтение и выполнение каталога, в котором находится файл. Например, если пользователь захочет ограничить доступ ко всем своим файлам, он может просто изменить права доступа своего домашнего каталога на «`drwx------`». Таким образом, никто другой не будет иметь доступ в его каталог (кроме «`root`»), а, следовательно, посторонним будут недоступны и все файлы – так что можно не заботиться об индивидуальной защите своих файлов. Другими словами, чтобы иметь доступ к файлу, необходимо иметь доступ ко всем каталогам, лежащим на пути от корня к этому файлу, а также разрешение на доступ собственно к этому файлу. При обращении к файлу или каталогу, доступа к которому у пользователя нет, ему будет выдано сообщение вроде «Permission denied» – «Отказано в доступе».

 *В символической ссылке не используются права доступа – они всегда отображаются, как «`rwxrwxrwx`». Вместо этого, права доступа к файлу, полученному символической ссылкой, определяются правами доступа к файлу, на который она ссылается.*

3.5. Интерфейс

Интерфейс служит для организации взаимодействия пользователя с компьютером. Управление работой ОС Linux осуществляется при помощи так

называемого *терминала*. Под терминалом (консолью) понимается устройство, предназначенное для обмена информацией между пользователем и компьютером. В минимальной конфигурации терминал состоит из монитора и клавиатуры, мышь – не обязательна. В ОС Linux существует два типа терминалов (интерфейсов):

3. *текстовый* (интерфейс командной строки – англ. Command Line Interface – *CLI*) и
4. *графический* (англ. Graphical User Interface – *GUI*).

В большинстве современных дистрибутивов Linux параметры настроены так, что по умолчанию первым загружается графический интерфейс (графический терминал).

Графический интерфейс пользователя (GUI) представляет собой разновидность пользовательского интерфейса, в котором реализована метафора *окружения рабочего стола* (англ. desktop environment). Рабочий стол в современных операционных системах – это экран, который видит пользователь сразу же после загрузки операционной системы. В нынешней компьютерной терминологии он является частью графической среды пользователя, которая, в свою очередь, призвана облегчить взаимодействие с компьютером, создав интуитивно понятный интерфейс, которым можно управлять как с помощью клавиатуры, так и мыши. «Интуитивно понятный интерфейс» достигается тем, что человек на экране видит графические изображения (кнопки, папки, документы, окна и др.), которыми он может управлять подобно тому, как он делал бы это с реальными объектами.

Графический интерфейс пользователя в ОС Linux строится но основе стандарта *X Window System* (заметьте, что Window, а не Windows – в единственном числе) или просто «*X*» (в просторечии – «*иксы*»). Первоначальный вариант этого стандарта был разработан в 1987 году в Массачусетском технологическом институте. Начиная со второй версии, стандарт поддерживается консорциумом X, созданным в 1988 г. с целью унификации графического интерфейса для ОС UNUX. С 1997 г. Консорциум X

преобразован в X Open Group. В настоящее время действует версия 11 выпуск 6 стандарта на графическую подсистему для UNIX-подобных ОС, который кратко обозначается как *X11R6*.

X Window System – это «многослойная» система, на самом «верху» которой находится *графическое окружение рабочего стола* (англ. *desktop environment*), которое, собственно, и является связующим звеном между конечным пользователем и операционной системой. Если в самой распространенной операционной системе Microsoft Windows, по сути, всего одна графическая оболочка, т. е пользователь может менять графическую тему, настройки некоторых графических элементов (например, изменить иконку у папки), но, как бы он не старался, сам графический интерфейс останется тем же. В ОС Linux ситуация кардинально противоположная – таких графических оболочек несколько. При этом пользователь имеет возможность установить их все, а при входе в систему выбрать ту, которая в данный момент ему больше подходит.

В ОС Linux наиболее распространенные графические среды это:

- *KDE* (K Desktop Environment или по-русски, просто, – окружение рабочего стола) и
- *GNOME* (GNU Network Object Model Environment или по-русски – сетевая среда объектной модели GNU).

Выбирая графическую среду, пользователь выбирает набор программ, с которыми он будете работать. Среда KDE использует для работы библиотеку Qt (The Q Toolkit), а GNOME – инструментарий GTK+ (The GIMP Toolkit). Из этого следует, что если пользователь выберет KDE, то будут установлены программы, работающие на библиотеке Qt, если же GNOME, то соответственно устанавливаются программы, основанные на GTK+. Например, в качестве файлового менеджера при выборе KDE будет установлен *Dolphin*, а при выборе GNOME – *Nautilus*. Но запустить прикладную программу под несвойственной ей средой не получится.

KDE – это графическая среда пользователя, по своему виду и логическому устройству, максимально приближена к GUI Microsoft Windows. Но это только внешне. Основной «изюминкой» KDE является гибкость в настройке системы, позволяющая конфигурировать приложения без правки текстовых файлов. Недостаток этой графической среды являются следствием ее преимущества – далеко не всем пользователям нужна такая гибкость и, соответственно, наличие такого большого количества опций для настройки. Для некоторых из них более важными являются именно те функциональные возможности, которые они предпочитают.

Первые версии KDE были созданы с использованием несвободных инструментов – библиотека Qt на тот момент была проприетарным продуктом. Это послужило причиной создания двух проектов:

- Harmony и
- GNOME.

Имея одинаковые цели (создание свободной среды свободными средствами), два проекта выбрали совершенно разные пути реализации задуманного. Проект Harmony ставил своей задачей переписать библиотеку Qt, выпустив ее под свободной лицензией, проект GNOME – полностью отказался от использования Qt. Последняя стабильная версия библиотеки Qt 4 доступна под лицензией GNU GPL для платформ UNIX, Mac OS, iOS, Android и Windows, что позволяет KDE иметь полную официальную поддержку на всех перечисленных платформах.

Разработка GNOME началась в августе 1997 года, как попытка создать полностью свободную рабочую среду для операционной системы Linux. Она была основана на инструментарии GTK+, созданном ранее для графического редактора GIMP и распространяемом на условиях GNU GPL. В GNOME основной приоритет получили соображения практичности, простоты и удобства использования среды, в том числе для неопытных пользователей. Ключевым моментом в GNOME является идея о том, что каждая функциональная нагрузка и каждая опция настройки в программе имеет свою цену: зачастую лучше

выбрать один, оптимальный вариант поведения программы, чем реализовывать множество вариантов и заставлять пользователя выбирать один из них.

Xfce является графической оболочкой, построенной на основе инструментального пакета GTK+, используемого в GNOME, но гораздо легче и предназначен для тех, кому нужен простой, эффективно работающий стол, который легко использовать и настраивать. Первый вариант Xfce был создан Оливером Форданом 1998 году. Название Xfce первоначально означало *XForms Common Environment*, но с того времени Xfce была переписана несколько раз и больше не использует эту технологию. (XForms – это технология Web-форм, которая основана на архитектуре *Model-View-Controller*, где данные представляются в виде XML.) Название осталось, но записывается уже не как XFce, а как Xfce и никак не расшифровывается.

С момента появления и до сегодняшнего дня Xfce позиционируется как GUI более «легкая» чем GNOME и KDE. Легкая в том смысле, что ей для работы требуется меньше оперативной памяти, более слабый процессор. То есть, по замыслу разработчиков она должна была своим быстродействием удовлетворить пользователей, у которых слабые компьютеры. В последнее время Линус Торвальдс тоже стал использовать Xfce, хотя и считает, что это «большой шаг назад по сравнению с GNOME 2, но это огромный шаг вперед по сравнению с GNOME 3».

LXDE (Lightweight X11 Desktop Environment) – это самая молодая GUI. Ее первый релиз был выпущен в 2006 году тайваньским программистом Хун Жень Йи. Она стремится предложить пользователю быстрый и легкий рабочий стол, нетребовательный к ресурсам компьютера и основанный на взаимно независимых компонентах. LXDE часто позиционируется как графическая среда еще более быстрая и «легкая» чем Xfce. Фактически в «легкости» разницы между ними практически нет, притом, что по функциональности она значительно уступает Xfce.

Unity является оболочкой рабочего стола разрабатываемая компанией Canonical для операционной системы Ubuntu Linux, первый выпуск которой

был 3 июня 2010 года. Среда рабочего стола Unity являющаяся ответвлением (англ. fork – ответвление) от кодовой базы GNOME 3. Она позволяет более эффективно использовать маленькие экраны нетбуков благодаря, например, вертикальной панели для переключения между запущенными программами. Все стандартные приложения Unity по-прежнему берутся из GNOME.

MATE – это среда рабочего стола, являющаяся ответвлением от кодовой базы более неподдерживаемой среды GNOME 2. Название MATE происходит от испанского названия (исп. mate) вида падуба, растения, из листьев которого готовят одноименный напиток. Одна из целей разработки MATE – сохранить работоспособность рабочего стола на старом оборудовании, не ориентируясь исключительно на современные конфигурации оборудования. Впервые о начале разработки MATE было объявлено 18 июня 2011 года на форуме Arch Linux пользователем Perberos, который и стал основателем проекта. Позднее к разработке MATE присоединились множество разработчиков и добровольных помощников.

Cinnamon (от англ. cinnamon – корица) – это среда рабочего стола, являющаяся ответвлением от кодовой базы GNOME. Основное направление разработки – предоставление пользователю более привычной среды в стиле GNOME 2, удобной пользователям настольных ПК и ноутбуков, без недостатков Unity. Изначально Cinnamon разрабатывался командой программистов Linux Mint. Проект впервые был представлен публике 2 января 2012 года.

Общение пользователя с операционной системой Linux в текстовом режиме (CLI) происходит через так называемую «*командную оболочку*» («*оболочку командной строки*», «*оболочку*» или, просто, «*shell*»). В терминах других операционных систем эта компонента называется еще «*интерпретатором команд*» или «*командным процессором*». Но в Linux (а также, других UNIX-подобных ОС) не совсем благозвучное название «*оболочка*» очень точно отражает суть проблемы – пользователь работает не с ОС непосредственно, а с ОС через специальную программу-оболочку – shell.

Т. е., все, что вводит пользователь на виртуальном терминале, вначале обрабатывается оболочкой. Далее, она вызывает на выполнение необходимые программы, а результаты их работы возвращает, опять же, пользователю на виртуальный терминал.

Имя программного файла одной из первых оболочек операционной системы UNIX было «sh» (сокращение от shell). Потом было разработано несколько ее улучшенных вариантов. В частности, *Bourne shell* – расширенная версия sh, написанная Стивом Борном (Steve Bourne). В рамках проекта GNU Брайеном Фоксом (Brain Fox – основной разработчик) и Четом Рэми (Chet Ramey) была разработана оболочка bash (от англ. «Bourne again shell» – «снова оболочка Борна» или «заново рожденная shell»).

На сегодняшний день разнообразие названий командных оболочек в мире UNIX может привести в замешательство кого угодно. Но, к этому нужно относиться как к результату эволюционного развития. На самом деле, существует только две основные группы (два основных типа) оболочек, это:

4. *Bourne shell* и
5. *C shell*.

Самая новая из командных оболочек, совместимая с оболочкой Борна – это *оболочка Z* (zsh). Оболочки группы C shell (csh и tcsh) используют синтаксис, чем-то напоминающий синтаксис языка программирования Си.

Выбор типа оболочки – это почти как выбор религии. Кто-то предпочитает синтаксис shell Борна, а кто-то – более структурированный синтаксис C shell. Однако, с точки зрения рядового пользователя, различия между всеми этими оболочками практически отсутствуют. Такие команды, например, как «ср и «ls», работают совершенно одинаково во всех типах оболочек. Различия начинают проявляться только при написании командных файлов, или использовании некоторых новых свойств оболочек. Однако, если есть такая возможность, попробуйте поработать с несколькими оболочками, прежде чем остановиться на одной из них. Это будет полезно на тот случай, если когда-нибудь Вы столкнетесь с системой, где выбор оболочек будет ограничен.

Самой же распространенной оболочкой, на сегодняшний день, в большинстве дистрибутивов Linux является bash. А термином shell обозначают, вообще, любую командную оболочку, в любой UNIX-подобной ОС.

3.6. Утилиты

Утилиты – это просто отдельные программы, которые выполняют некоторые служебные функции, например – копирование или удаление файлов. Большинство утилит в ОС Linux расположены в каталогах `/bin` и `/usr/bin`. Полный же список каталогов, где командная оболочка ищет утилиты, называется «*путь поиска*». Он хранится в переменной окружения `PATH`. Посмотреть ее текущее значение можно по команде «`echo $PATH`». Результатом работы команды будет последовательность путей, разделенных символом двоеточия («`:`»). А команда «`set`» без параметров выдает текущие значения всех переменных окружения.

3.7. Дистрибутивы

Общее название Linux не подразумевает какой-либо единой «официальной» комплектации. Она распространяется в основном бесплатно в виде различных готовых «дистрибутивов», имеющих свой набор прикладных программ и уже настроенных под конкретные нужды пользователя. И в то же время, как одно из следствий свободного распространения ПО для Linux является то, что большое количество различных фирм и компаний, а также просто независимых групп разработчиков стали выпускать так называемые «дистрибутивы Linux». Дистрибутив – это набор программного обеспечения, включающий все 4 основные составляющие ОС, т. е. ядро, файловую систему, интерфейс и утилиты, а также некоторую совокупность прикладных программ. Обычно все программы, включаемые в дистрибутив Linux, распространяются на условиях GPL. Однако разработчик дистрибутива должен, по крайней мере, укомплектовать его программой инсталляции, которая будет устанавливать Linux на компьютер, на котором никакой ОС еще нет. Кроме того, необходимо обеспечить разрешение взаимозависимостей и противоречий между разными *пакетами программ*, а также их версиями. На сегодняшний день в мире

существует уже более сотни различных дистрибутивов Linux, и все время появляются новые. Например, это: Ubuntu Linux, Linux Mint, Debian Linux, Red Hat Linux, Fedora Linux, ASPLinux, Gentoo Linux, Arch Linux, ALT Linux, openSUSE Linux, PCLinuxOS и др.

4. Полиморфизм

Полиморфизм (*polymorphism*) – это свойство компьютерных программ, которое позволяет использовать одни и те же приемы и методы для решения нескольких схожих, но технически различных задач. В общем виде идея полиморфизма может быть коротко сформулирована следующим образом – «один интерфейс – и множество реализаций».

Принцип полиморфизма не является чем-либо совершенно новым, или «революционным изобретением» программистов, – он уже давно и широко используется в различных областях науки и техники. Так, например, последовательность расположения педалей сцепления, тормоза и газа совершенно одинаков во всех автомобилях, в не зависимости от типа их двигателя (карбюраторный, или дизельный), области применения (легковой или грузовой), класса, марки, места производства и т. д. Это позволяет водителям совершенно легко и просто «пересаживаться» с одного автомобиля на другой. С другой стороны, не составит особого труда представить себе «коммерческий успех» автомобиля, у которого конструкторы «оптимизировали» и, соответственно, изменили (без веских на то оснований) порядок расположения этих педалей! Единственное, что сделали программисты, так это то, что оно дали подходящее название одному из самых существенных признаков своего товара.

В программировании полиморфизм достаточно широко используется при создании самых различных программ, самого разнообразного назначения. При этом, однажды освоив некоторый набор приемов работы с одним приложением, затем большинство из них можно использовать и при работе с другими приложениями. Естественно, что в каждом конкретном приложении, и его

состоянии, могут быть задействованы совершенно разные механизмы. Так, например, перемещение или копирование выделенного фрагмента в пределах одного документа реализуются с помощью элементарных функций перемещения и копирования конкретного приложения. Те же действия, но между различными документами одного приложения, могут задействовать совсем другие механизмы (например, технологию DDE – Dynamic Data Exchange в Microsoft Windows), а между различными приложениями – третьи (например, технологию OLE – Object Linking and Embedding в Microsoft Windows). Однако, с точки зрения пользователя, все эти «тонкости реализации» никакого значения не имеют – для него существенным является лишь то, что чтобы одинаковые действия в различных приложениях приводили бы к одинаковым результатам.

Следование принципам полиморфизма, как и любым стандартам, вообще, имеет как свои сильные, так и слабые стороны. Так, в науке любые стандарты являются своеобразным «тормозом» для достижения новых результатов. В промышленности же, при производстве товаров, например, следование стандартам является необходимостью, поскольку предоставляет много дополнительных преимуществ. В частности, при массовом производстве использование унифицированных технологий всегда является более дешевым. Пользователи также всегда отдают предпочтение тем товарам, которые им давно и хорошо известны, и совсем неохотно «осваивают» совершенно новые изделия – все люди немножечко ленивы, и не любят «переучиваться».

А поскольку, программы также является товаром, то, естественно, что большинство разработчиков последовательно и целенаправленно придерживаются идеи полиморфизма. Для пользователей же эта особенность означает лишь одно – что, однажды освоив интерфейс и приемы работы с одной из программ, затем, их можно использовать и при работе с другими приложениями. Поэтому, данную методику следует освоить как можно раньше, и как можно лучше, поскольку ею придется пользоваться постоянно и повсеместно.

ЛЕКЦИЯ № 4

СЕТЕВЫЕ ТЕХНОЛОГИИ

План

1. Компьютерные сети
2. Простейшая компьютерная сеть
3. Классификация компьютерных сетей
4. Топологии компьютерных сетей
5. Модель взаимодействия открытых систем ISO/OSI
6. Понятие протокола и интерфейса

1. Компьютерные сети

Компьютерная сеть – это совокупность компьютеров, между которыми возможен прямой обмен информацией без использования промежуточных носителей. Для создания компьютерной сети необходимо выполнение двух условий:

1. чтобы входящие в нее компьютеры были соединены между собой *каналами передачи информации* (*каналами связи*), и
2. чтобы на каждом из них было установлено *специальное программное обеспечение для поддержания работы сети* (*программы управления сетью*).

При этом технология работы в сети зависит как от способа организации каналов связи, так и от программного обеспечения.

2. Простейшая компьютерная сеть

Она образуется при соединении двух компьютеров в пределах нескольких метров при помощи специального кабеля. Этот кабель называется *нуль-модемном* и подключается к последовательным (COM) или параллельным (LPT) портам обеих компьютеров. Такое соединение компьютеров, как правило, является временным и называется *прямым компьютерным*

соединением. Оно может быть установлено, а затем снято любым конечным пользователем. Для каждого вида соединения (через COM или LPT-порты) необходим свой вид кабеля. При этом в случае соединения компьютеров через параллельные (LPT) порты обмен информацией выполняется $2 \div 4$ раза быстрее.

⚠ *Нуль-модемный кабель присоединяется при выключенных компьютерах!*

Прямые компьютерные соединения используются в основном для обмена информацией между портативными и стационарными офисными компьютером, хотя такой обмен возможен и между двумя стационарными компьютерами.

При прямом компьютерном соединении возможны два типа их взаимодействия:

- *прямой доступ*, при котором возможна только пересылка информации с одного компьютера на другой и
- *удаленное управление*, при котором возможно выполнение программ, размещенных на другом компьютере.

Операционная система Microsoft Windows внутренними средствами удаленного управления для прямого кабельного соединения не обладает.

В последнее время для обмена информацией между двумя или несколькими компьютерами все более широкое распространение получают *беспроводные* (*wireless*) способы соединения, в которых используются электромагнитные волны *инфракрасного* (*IrDA*) и *радиочастотного* (*Blue Tooth*, *Wi-Fi*) диапазонов. При этом способы построения таких сетей упрощаются, а функциональные возможности – повышаются.

3. Классификация компьютерных сетей

В зависимости от компьютеров, каналов связи, установленного программного обеспечения и некоторых других признаков компьютерные сети принято классифицировать по:

- типу среды передачи данных;
- способу организации взаимодействия компьютеров;

- скорости передачи данных;
- занимаемой территории;
- ведомственной принадлежности;
- топологии.

В зависимости от **типа среды передачи данных** компьютерные сети делятся на:

- *проводные*,
- *беспроводные* и
- *смешанные* или *гибридные*, в которых одни фрагменты – проводные, а другие – беспроводные.

По **способу организации взаимодействия компьютеров** различают два типа сетей:

- *одноранговые сети* и
- *сети с выделенным сервером* (типа *клиент/сервер*).

Одноранговые сети не предусматривают выделение специальных компьютеров, организующих работу сети. Каждый пользователь, подключаясь к сети, выделяет в сеть какие-либо ресурсы (дисковое пространство, принтеры) и подключается к ресурсам, предоставленным в сеть другими пользователями. Такие сети просты в установке и налаживании; они существенно дешевле сетей с выделенным сервером. В свою очередь, **сети с выделенным сервером**, несмотря на сложность настройки и относительную дороговизну, позволяют осуществлять централизованное управление.

По **скорости передачи данных** сети делятся на:

- *низко*-,
- *средне*- и
- *высокоскоростные*.

Основной критерий разделения – скорость передачи данных. Понятно, что скорость передачи данных – понятие не постоянное. То, что сегодня считается среднескоростным соединением, завтра будет отнесено к низкоскоростным. Тем не менее, сегодня низкоскоростной сетью считается сеть со скоростью

передачи информации до 10 Мбит/с. Среднескоростная сеть передает данные со скоростью до 100 Мбит/с, а высокоскоростные сети передают информацию со скоростью свыше 100 Мбит/с.

По **занимаемой территории** сети могут быть:

- *локальными*,
- *региональными* (*муниципальными*) и
- *глобальными*.

Локальная вычислительная сеть (*ЛВС*) или по-английски *Local Area Network – LAN*, объединяет компьютеры, расположенные в пределах одного помещения, внутри одного или нескольких зданий, между которыми необходимо организовать постоянный обмен информацией. Такие сети создаются при помощи специального стационарного сетевого оборудования.

Региональная вычислительная сеть по-английски называется *Metropolitan Area Network – MAN* и по своему функциональному назначению такая сеть эквивалентна локальной вычислительной сети. Она, как правило, объединяет в единое целое несколько сетей значительно удаленных друг от друга в пределах одного населенного пункта, или одного региона, между которыми необходимо организовать постоянный обмен большими потоками информации. Компьютеры в такой сети соединяются при помощи постоянно действующих выделенных каналов связи. При этом среда передачи данных сети MAN может быть как проводной, так и беспроводной. Сети, связывающие значительно удаленные друг от друга компьютеры, так же называются *распределенными*.

Глобальная сеть по-английски называется *Wide Area Network – WAN*. Она объединяет компьютеры, расположенные по всему миру, на которых имеются огромные объемы разнообразной информации и доступной всем желающим как свободно, так и на коммерческой основе. Компьютеры в такой сети связанные каналами с очень высокой пропускной способностью. Наиболее известной глобальной сетью является *Интернет*, хотя имеются и другие, например, *Microsoft on Line*, *America on Line*.

В зависимости от **прав собственности на сети** (**ведомственной принадлежности**) последние могут быть двух видов:

1. *общего пользования* (*public*) и
2. *частные* (*private*).

Среди сетей общего пользования выделяются телефонные сети (*PSTN – Public Switched Telephone Network*), сети передачи данных (*PSDN – Public Switched Data Network*) и высокоскоростные цифровые сети с предоставлением комплексных услуг (*ISDN – Integrated Services Digital Network*).

Информационные системы, в которых средства передачи данных принадлежат одной компании и используются только для нужд этой компании, принято называть *Сетью Масштаба Предприятия* или *Корпоративной Сетью* (*Enterprise Network*).

4. Топологии компьютерных сетей

При объединении в сеть более двух компьютеров всегда необходимо решать задачу, каким образом их соединить друг с другом. И при увеличении числа связываемых устройств, число возможных способов конфигураций резко возрастает. Каждый из таких вариантов физических связей образует конфигурацию или *топологию* сети.

Под топологией сети понимается граф, вершинам которого соответствуют компьютеры или другое коммуникационное оборудование, а ребрам – каналы связи между ними.

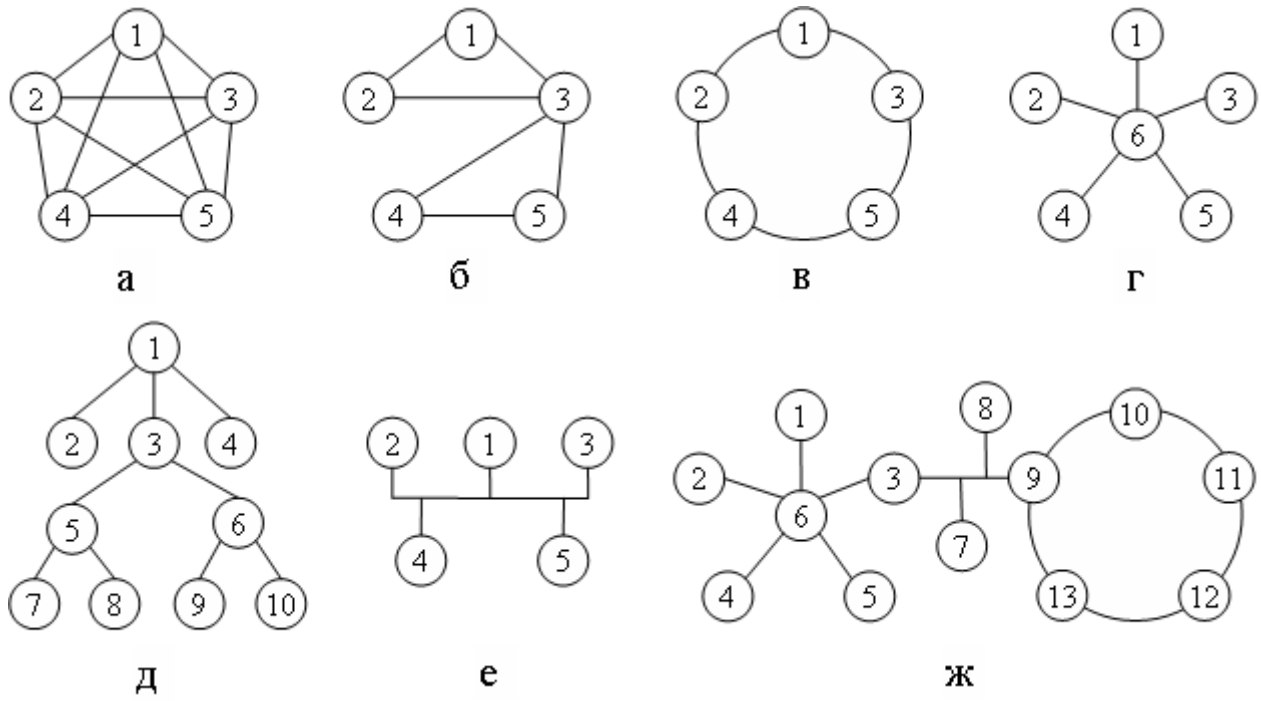
Среди всего множества конфигураций компьютерных сетей различают:

1. *полносвязные* и
2. *неполносвязные*.

Полнозвязная топология (рисунок 4.1, а) соответствует сети, в которой каждый компьютер непосредственно связан со всеми остальными. Это самая надежная топология сети. Но, несмотря на кажущуюся простоту, этот вариант оказывается громоздким и не эффективным потому, что каждый компьютер должен иметь достаточное количество коммуникационных портов для связи со

всеми остальными компьютерами сети. Поэтому практически ни когда не используется, поскольку является самым дорогим и трудным в обслуживании. Этот вид топологии применяется лишь в многомашинных комплексах или сетях, объединяющих небольшое количество компьютеров.

Все другие варианты (рисунок 4.1, б ÷ ж) основаны на *неполносвязных топологиях*, когда для обмена данными между двумя компьютерами может потребоваться транзитная передача данных через другие узлы сети.



а. Полносвязная
б. Ячеистая
в. Кольцевая (Ring)
г. Звездообразная (Star)
д. Иерархическая звезда (дерево)
е. Общая шина (Bus)
ж. Смешанная

Рисунок 4.1 – Типовые топологии компьютерных сетей

Ячеистая топология (рисунок 4.1, б) получается из полносвязной путем удаления некоторых связей. Она допускает соединения большого количества компьютеров и характерна, как правило, для крупных сетей. В такой сети есть, по крайней мере, два узла имеющих два или более пути между ними.

В сетях с *кольцевой топологией* (*Ring*) данные передаются по кольцу от одного компьютера к другому (рисунок 4.1, в). Главным достоинством такой топологии является то, что она по своей природе обеспечивает резервирование связей. Действительно, любая пара узлов соединена здесь двумя путями – по часовой стрелке и против нее. Кольцо представляет собой очень удобную

конфигурацию и для организации обратной связи – данные, сделав полный оборот, возвращаются к узлу источнику. Поэтому источник может контролировать процесс доставки информации потребителю. Часто это свойство кольца используется для тестирования связной сети и поиска неисправностей. В то же время в сетях с кольцевой топологией необходимо принимать специальные меры, чтобы в случае выхода из строя или отключения какого-либо компьютера не прерывался канал связи между остальными узлами кольца.

Звездообразная топология (*Star*) образуется, когда каждый компьютер подключается непосредственно к центральному устройству, которое называется *концентратором* (рисунок 4.1, г). В его функции входит направление передаваемой компьютером информации одному или всем остальным компьютерам сети. В качестве концентратора может выступать как универсальный компьютер, так и любое специализированное многовходовое устройство, такое, например, как *коммутатор* (англ. *switch*), маршрутизатор (англ. *router*) или тот же физический концентратор (hub). К недостаткам топологии типа звезда относится более высокая стоимость сетевого оборудования из-за необходимости приобретения специализированного центрального устройства. Кроме того, возможности по наращиванию количества узлов в сети ограничиваются количеством портов концентратора. Иногда имеет смысл строить сеть с использованием нескольких концентраторов, иерархически связанных между собой связями типа звезда (рисунок 4.1, д). Полученную в результате структуру называют *иерархической звездой* или *деревом*. В настоящее время дерево является самой распространенной топологией связей, как в локальных, так и в глобальных сетях.

Конфигурация *общая шина* (*Bus*) является особым частным случаем звезды (рисунок 4.1, е). В такой сети все узлы подключены к единой среде передачи данных, например, к коаксиальному кабелю. Аналогичную топологию имеют многие сети, использующие беспроводную связь, в которых в качестве общей

шины выступает общий радиоканал. Передаваемая информация распространяется по шине и одновременно доступна всем компьютерам, присоединенным к ней. Основными преимуществами такой схемы являются дешевизна и простота присоединения новых узлов к сети, а недостатками – низкая надежность (любой дефект шины полностью парализует всю сеть) и невысокая производительность (в каждый момент времени только один компьютер может передавать данные по сети, поэтому пропускная способность делится между всеми узлами сети).

В то время как небольшие сети, как правило, имеют типовую топологию такую, например, как звезда, кольцо или общая шина, для крупных сетей характерно наличие произвольных связей между компьютерами. В таких сетях можно выделить лишь отдельные фрагменты (подсети), имеющие типовую топологию. Поэтому такие сети называются сетями со *смешанной топологией* (рисунок 4.1, ж).

5. Модель взаимодействия открытых систем ISO/OSI

К концу 70-х годов XX столетия в мире уже существовало большое количество всевозможных компьютерных сетей, построенных по самым разнообразным технологиям. Такое разнообразие сетевых технологий вывело на первый план проблему обмена информацией между ними. Решение этой проблемы относится к области стандартизации. Оно было получено в 80-е годы, кода Международная организация по стандартизации (*ISO – International Standards Organization*) разработала, а сообщество сетевых проектировщиков приняло, стандартную *модель взаимодействия открытых систем* (*OSI – Open System Interconnection*).

Основная идея модели OSI заключается в том, что общая задача передачи данных расчленяется на отдельные, легкообозримые части (или уровни). Она разрабатывалась в качестве своего рода универсального языка сетевых спецификаций и сыграла значительную роль в развитии компьютерных сетей. Именно поэтому ее еще называют *справочной моделью*, *семиуровневой моделью*

OSI или, просто, *моделью OSI*. Не смотря на то, что модель OSI сугубо теоретическая, на практике она оказалась исключительно полезной – без нее в сетях воцарился бы полный хаос, когда оборудование от разных производителей не смогло бы работать вместе. Модель OSI позволяет, в случае необходимости, изменить или заменить какой-либо отдельный элемент или составляющую сети без изменения всей сети. При этом также не требуется перепроектирование всей сети с нуля.

Модель OSI определяет:

1. *уровни взаимодействия в компьютерных сетях,*
2. *стандартные названия уровней и*
3. *функции, которые должен выполнять каждый уровень.*

Модель OSI не содержит описаний реализаций конкретного набора правил сетевого взаимодействия.

В модели OSI (рисунок 4.2) средства сетевого взаимодействия делятся на семь уровней:

1. физический,
2. канальный,
3. сетевой,
4. транспортный,
5. сеансовый,
6. представления и
7. прикладной.

Каждый из них имеет дело с совершенно определенным аспектом сетевых взаимодействия.

Пусть, например, приложению **А**, выполняющемуся на компьютере №1, необходимо переслать информацию приложению **Б** на компьютер №2. Для этого приложение **А** через интерфейс прикладного программирования (англ. *API – Application Program Interface*) обращается с запросом к прикладному уровню. На основании этого запроса программное обеспечение прикладного уровня формирует сообщение стандартного формата. После формирования

сообщения прикладной уровень направляет его вниз на уровень представления. После выполнения необходимых действий на этом уровне сообщение передается на следующий уровень вниз и т.д., до достижения физического уровня.

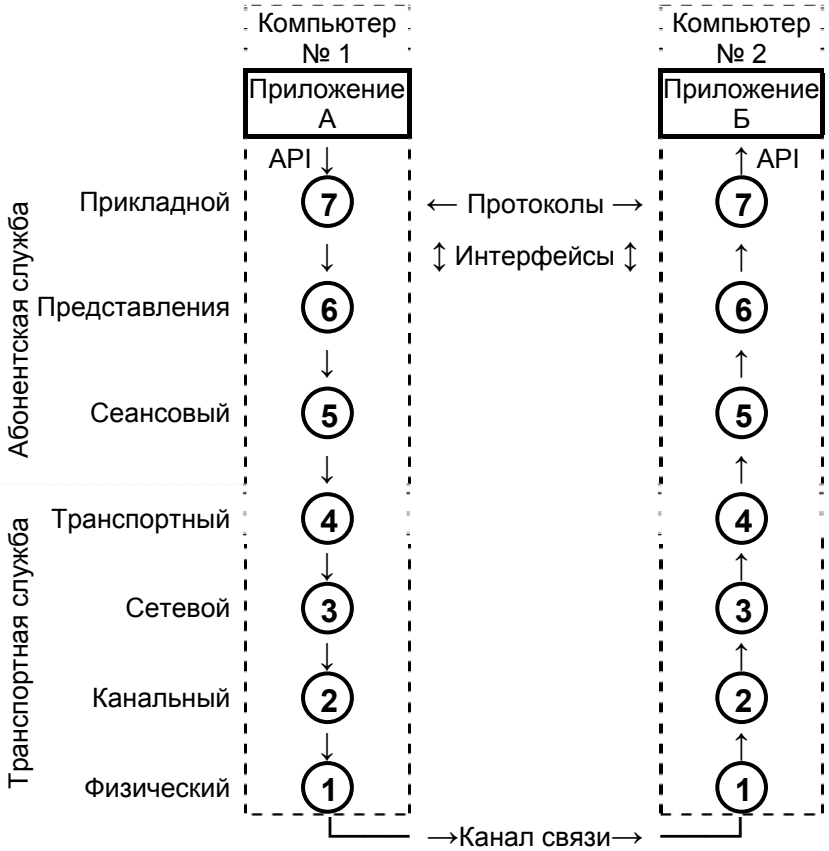


Рисунок 4.2 – Модель взаимодействия открытых систем ISO/OSI

На физическом уровне сообщение передается на выходной интерфейс компьютера №1, и далее оно начинает свое «путешествие» по сети. До этого сообщение передавалось от одного уровня к другому в пределах компьютера №1. Когда сообщение по сети поступает на входной интерфейс компьютера №2, оно принимается его физическим уровнем и последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует и обрабатывает сообщение, выполняя при этом все необходимые действия, и передает его вышележащему уровню. В результате информация, переданная приложением **А** из компьютера №1, будет получена приложением **Б** на компьютере №2.

Рассмотренная схема взаимодействия приложений соответствует идеальной модели OSI, когда прикладная программа обращается с запросом к

самому верхнему уровню – прикладному. Однако на практике во многих случаях имеется возможность прикладным программам напрямую обращаться к нижележащим уровням. Например, некоторые Системы Управления Базами Данных (СУБД) имеют встроенные средства удаленного доступа к файлам. В этом случае приложение, выполняя доступ к удаленной информации, не использует средства файловой системы. Оно обходит верхние уровни модели OSI и непосредственно обращается к ответственным за транспортировку средствам, которые располагаются на более низких уровнях модели OSI.

Физический уровень (*physical layer*) имеет дело с передачей потока битов по физически каналам связи, таким, например, как кабели или беспроводные каналы передачи информации. Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптерам или одним из портов. Физический уровень не вникает в смысл информации, которую он передает. Для него эта информация представляется однородным потоком битов, которую нужно доставить адресату без искажений.

Канальный уровень (*data link layer*). В некоторых сетях, в которых линии связи используются попеременно (разделяются) несколькими парами взаимодействующих компьютеров, физическая среда передачи данных может быть занята. Поэтому одной из задач канального уровня является проверка доступности среды передачи. Другой задачей канального уровня является реализация механизмов обнаружения и исправления ошибок. Для этого на канальном уровне биты группируются в наборы, называемые *кадрами* (*frames*). Канальный уровень обеспечивает корректность передачи кадров, помещая в его начало и конец специальную последовательность бит – контрольную сумму.

Функции средств канального уровня определяются по-разному для локальных и глобальных сетей:

- В *локальных сетях* канальный уровень должен обеспечить доставку кадра между *любыми* узлами сети.

- В *глобальных сетях* канальный уровень должен обеспечить доставку кадра только между двумя *соседними* узлами.

Топология сети (шина, звезда и т. д.) определяется как раз на канальном уровне. На этом уровне также определяется работа с так называемыми MAC-адресами. *MAC-адрес* – это уникальный аппаратный адрес сетевого устройства, например, сетевого адаптера, точки доступа и т. д. Каждому производителю сетевых устройств выделяется свой диапазон MAC-адресов. В мире не существует двух сетевых устройств с одинаковыми MAC-адресами.

Сетевой уровень (*network layer*) служит для образования единой транспортной системы, объединяющей несколько сетей. Причем эти сети могут использовать совершенно различные принципы передачи сообщений и обладать произвольной структурой.

 *Технология, позволяющая соединить в единую сеть множество сетей, в общем случае построенных на основе разных технологий, называется технологией* ***межсетевого взаимодействия (internetworking)****.*

Функции сетевого уровня реализуются:

1. группой протоколов и
2. специальными устройствами – маршрутизаторами.

Маршрутизатор – это устройство, которое соединяет между собой различные сети. Он собирает информацию о топологии межсетевых соединений и на ее основе пересылает сообщения. Такие сообщения на сетевом уровне называются *пакетам* (*packets*). Для того чтобы передать пакет от отправителя, находящегося в одной сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество *транзитных передач между сетями*. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет. Единица такого перехода называется *хопом* (от англ. *hop – прыжок*). Количество хопов равно количеству маршрутизаторов между двумя сетями. Проблема выбора наилучшего пути называется *маршрутизацией*, и ее решение является одной из главных задач сетевого уровня.

Маршрутизатор может быть реализован программно, на базе универсального компьютера, работающего под управлением операционной системы UNIX или Windows. Однако чаще маршрутизаторы реализуются на базе специализированных аппаратных платформ. В состав программного обеспечения маршрутизатора входят протокольные модули сетевого уровня.

Примерами протоколов сетевого уровня являются протокол межсетевого взаимодействия IP стека TCP/IP и протокол обмена пакетами IPX стека Novell.

Транспортный уровень (*transport layer*) обеспечивает верхним уровням модели OSI – прикладному, представления и сеансовому – передачу данных с той степенью надежности, которая им требуется. То есть он отвечает за обнаружение и исправление таких ошибок, как искажение, потеря и дублирование пакетов.

Выбор глубины и качества обнаружения и исправления ошибок передачи данных определяется, с одной стороны, тем, в какой степени задача обеспечения надежности решается самими приложениями и протоколами более высоких уровней. С другой стороны этот выбор зависит от того, насколько надежной является система транспортировки данных в сети, обеспечиваемая более низкими уровнями – сетевым, канальным и физическим. Так, если качество каналов связи высоко и вероятность возникновения ошибок невелика, то разумно воспользоваться одним из облегченных вариантов. Если же транспортные средства нижних уровней не очень надежны, то целесообразно использовать максимум средств обнаружения и устранения ошибок.

Все протоколы, начиная с транспортного уровня и выше, реализуются программными средствами конечных узлов сети – компонентами их сетевых операционных систем. В качестве примера транспортных протоколов можно привести протокол TCP стека TCP/IP или SPX стека Novell.

Сеансовый уровень (*session layer*) обеспечивает управление диалогом для того, чтобы фиксировать, какая из сторон является активной в настоящий момент, а также предоставлять средства синхронизации сеанса. Эти средства синхронизации позволяют в ходе длительных передач данных сохранять

информацию о состоянии этих передач в виде контрольных точек, чтобы в случае отказа можно было вернуться назад к последней из них, а не начинать все с начала. На практике немногие приложения используют сеансовый уровень, и он редко реализуется в виде отдельных протоколов. Функции этого уровня часто объединяются с функциями прикладного уровня и реализуются в одном протоколе.

Уровень представления (*presentation layer*), как явствует из его названия, имеет дело с формой представление передаваемой по сети информации, не меняя при этом ее содержание. За счет уровня представления информация, передаваемая прикладным уровнем одной системы, всегда понятна прикладному уровню другой системы. С помощью средств данного уровня протоколы прикладных уровней могут преодолевать синтаксические различия в представлении данных или же различия в их кодировках. Например, кодов символов ASCII и EBCDIC. На этом уровне может выполняться шифрование и дешифрование информации, благодаря которым секретность обмена данными обеспечивается сразу для всех прикладных служб. Примером такого протокола является *слой защищенных сокет* (*SSL – Secure Socket Layer*), который обеспечивает секретность обмена сообщениями для протоколов прикладного уровня стека TCP/IP.

Прикладной уровень (*application layer*) – это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например, по протоколу электронной почты. Единица данных, которой оперируют на прикладном уровне, обычно называется *сообщением* (*message*).

Существует очень большое разнообразие протоколов и соответствующих служб прикладного уровня. Например, наиболее распространенные реализации сетевых файловых служб являются NCP в операционной системе Novell NetWare, SMB в Microsoft Windows, NFS в стеке TCP/IP. На этом уровне работает множество самых разнообразных протоколов, например, HTTP (Hyper

Text Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) и т. д.

Четыре нижних уровня стандартной модели OSI образуют *транспортную службу* компьютерной сети, которая полностью решают задачу транспортировки сообщений с заданным уровнем качества, освобождая более высокие уровни от решения этих задач. В свою очередь, три верхних уровня, обеспечивающие логическое взаимодействие прикладных процессов, функционально объединяются в *абонентскую службу*.

6. Понятие протокола и интерфейса

В сущности, термин «протокол» и «интерфейс» выражают одно и то же понятие – формализованное описание процедуры взаимодействия двух объектов. Но традиционно в сетях за ними закрепились разные области действий. *Протоколы* определяет правила взаимодействия модулей одного уровня в разных узлах (*по горизонтали*), а *интерфейсы* – правила взаимодействия модулей соседних уровней в одном узле (*по вертикали*).

ЛЕКЦИЯ № 5

ПРИМЕНЕНИЕ ИНТЕРНЕТА В ЭКОНОМИКЕ

План

1. История создания Интернета
2. Структура и принципы работы Интернета
3. Адресация компьютеров в сети
4. Доменная система имен
   - Хост-файл
   - Создание DNS
   - Пространство доменных имен
   - Механизм разрешения
   - Регистрация доменов
   - dotCOM-бизнес
5. Единообразный локатор ресурса

1. История создания Интернета

Непосредственной предшественницей Интернета была компьютерная сеть *APRANET*, созданная в 1969 году рядом научных и исследовательских организаций под контролем одного из подразделений Министерства обороны США – агентства *ARPA* (Advanced Research Projects Agency – Агентство Передовых Исследовательских Проектов). В разные годы своего существования агентство несколько раз меняло название. Так, с момента создания в 1958 г. оно называлось ARPA; затем в 1972 г было переименовано в DARPA (с добавлением слова Defense – Оборона); затем в 1993 – опять в ARPA, и, наконец, 11 марта 1996 года – снова в DARPA. Основная цель, которая ставилась при создании сети, состояла в том, что она должна выполнять свои функции даже при частичном ее разрушении. Например, в результате военных

действий, пожара, стихийного бедствия или простого воровства оборудования. Это был *эксперимент по пакетной коммутации сообщений*.

Первоначально сеть ARPANET состояла из 4-х узлов (рисунок 5.1):

1. компьютер SDS SIGMA в Калифорнийском университете Лос-Анджелеса, UCLA (внизу),
2. компьютер SDS940 в Стэндфордском исследовательском институте, SRI (вверху в центре),
3. компьютер IBM360 в Калифорнийском университете в городе Санта-Барбара, UCSB (слева),
4. компьютер DEC PDP-10 в университет штата Юта в Солт-Лейк Сити, UTAH (справа).

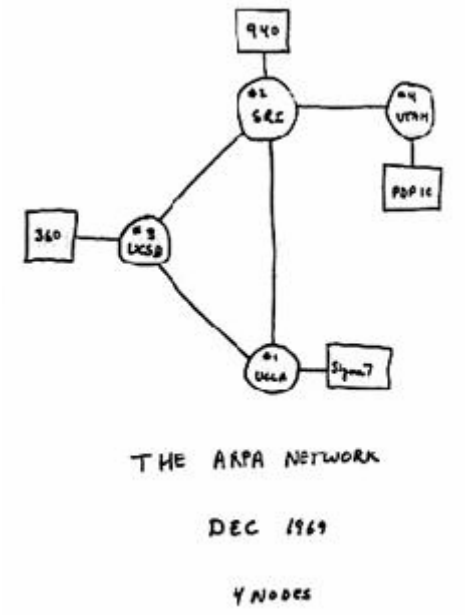


Рисунок 5.1 – Первоначальный вариант сети ARPANET

В сети ARPANET все узлы (компьютеры) имеют одинаковый статус. При этом каждый из них уполномочен порождать, передавать и получать сообщения от любого другого. Сообщения для передачи разбиваются на небольшие стандартизированные элементы, называемые *пакетами*. Каждый такой пакет имеет адрес назначения, и доставка сообщения обеспечивается тем, что каждый узел имеет возможность посылать (или переадресовывать) пакеты по сети к месту назначения. Компьютеры в сеть объединялись при помощи разработанных в BBN – Bolt, Beranek & Newman (Кембридж, шт. Массачусетс),

устройств *IMP* (*Interface Message Processor – Интерфейса Процессора Сообщений*), размером с холодильник, которые осуществляли связи между компьютерами через *телефонную сеть*.

Первые испытаний ARPANET оказались крайне успешными. Ученые исследовательских учреждений, послуживших испытательными полигонами, получили возможность передавать данные и совместно пользоваться удаленным доступом к компьютерам. К лету 1970-го года сеть насчитывала уже целых 10 узлов, а через год их стало уже 15. Университет Гавайи обзавелся собственной сетью, напоминающей ARPANET – *ALONANET*. С каждым годом ARPANET росла и развивалась – в нее включались все новые и новые участники. Право доступа в сеть начали требовать себе сначала все крупные лаборатории, потом – и более мелкие. Наконец, в гонку за ARPANET включились и высшие учебные заведения. Военные ворчали, но соглашались. В 1972 году сеть уже связывала 50 университетов и исследовательских центров.

Одновременно с ростом сети велись работы по созданию функционально полного протокола межкомпьютерного взаимодействия и другого сетевого программного обеспечения. В декабре 1970 года Сетевая рабочая группа (Network Working Group – NWG) под руководством С. Крокера завершила работу над первой версией протокола, получившего название *Протокол управления сетью* (*Network Control Protocol – NCP*). После того, как в 1971 – 1972 годах были выполнены работы по реализации NCP на узлах ARPANET, пользователи сети, наконец, смогли приступить к разработке приложений.

В 1972 году появилось первое «сетевое» приложение – *электронная почта*. В марте Рэй Томлинсон (Ray Tomlinson) из BBN, движимый необходимостью создания для разработчиков ARPANET простых средств координации, написал базовые программы пересылки и чтения электронных сообщений. Он же предложил использовать значок «@», который и по сей день является неотъемлемой частью любого электронного адреса. Интересно, что в разных странах его называют совершенно по разному: у нас – «собачка», в Дании – «хобот слона», в Греции – «маленькая утка», а в Германии – «висящая

обезьяна». Чего-чего, а чувства юмора программистам всего мира занимать не приходится. В июле того же года Лоуренс Робертс добавил к этим программам возможности выдачи списка сообщений, выборочного чтения, сохранения в файле, пересылки и подготовки ответа. Совместимость различных почтовых систем, работающих на разных компьютерных платформах, продемонстрировала выгоды массовых электронных коммуникаций между людьми. С тех пор более чем на десять лет электронная почта стала крупнейшим сетевым приложением. Для своего времени электронная почта стала тем же, чем в наши дни является Всемирная Паутина – исключительно мощным катализатором роста всех видов межперсональных потоков данных.

В октябре 1972 решено было провести Международную конференцию по компьютерным коммуникациям (International Computer Communication Conference – ICCC). Идея была в том, чтобы установить временный интерфейсный процессор (TIP – Terminal Interface Processor) в здании Washington Hilton Hotel и дать публике войти и использовать ARPANET, выполняя приложения по всем Штатам. Готовились почти год. В подготовку к демонстрации вовлекли лучших специалистов из ARPA и BBN. Боб Кан (Bob Kahn) из BBN в течение трех дней весьма успешно демонстрировал работу сети. Профессионалы не хотели, а может быть, и не могли поверить, что сеть, содержащая 100 компьютеров, может надежно работать. Это был первый показ на публике новой сетевой технологии.

После конференции наступает новый этап в развитии ARPANET. В 1973 году впервые были подключены зарубежные узлы – Университетский колледж в Лондоне (University College of London) и Королевская лаборатория радиолокации в Норвегии (Rogee Radar Establishment). Тогда же была запущена спутниковая линия связи с Гавайским университетом. С этого момента ARPANET становится международной и основной задачей становится объединение разнородных сетей в единую сеть.

В середине 1975 года DARPA, пришло к выводу, что ее сеть стабильна, и управление ARPANET было передано DCA – Defense Communications Agency –

Оборонному Агентству по Коммуникациям США (в настоящее время оно называется DISA – Defense Information Systems Agency – Оборонное Агентство по Информационным Системам). Так ARPANET превратилась из экспериментальной сети в рабочую сеть.

Тем не менее, DARPA (и не только оно) продолжало заниматься техническими аспектами ARPANET. Одним из таких направлений была разработка межсетевых протоколов, так как PP-протоколы (Point-to-Point – протоколы типа Точка-Точка) уже не могли обеспечивать подключения такого большого количества различных по структуре локальных сетей, желающих подключиться к ARPANET. Это было обусловлено спецификой технической реализации сети, которая состояла в том, что основой ARPANET являлись соединенные между собой IMP-узлы. С течением времени IMP-узлы были переименованы в *PSN*-узлы (*Packet Switch Node – Узлы Коммутации Пакетов*) и само оборудование, представляющее собой эти устройства IMP/PSN, было модернизировано. PSN-узлы были связаны между собой каналами связи типа точка-точка. Причем, связаны они так, чтобы каждый PSN-узел имел, как минимум, два канала связи с двумя разными PSN-узлами. При таком положении, если откажет один канал связи или один PSN-узел, связь в сети ARPANET не будет нарушена, так как другие PSN-узлы смогут отправить свои пакеты в обход аварийного участка. Каждый PSN-узел укомплектован двадцатью двумя внешними портами, к которым можно подключать клиентские машины. Машина, подключенная к порту PSN-узла, называется *хостом*. Обмен данными между хостом и PSN-узлом происходил по протоколу *X.25*. Для идентификации машины в сети ARPANET использовалась следующая схема адресации: каждый PSN-узел получает свой уникальный номер, а так как каждый порт PSN-узла тоже имеет конкретный номер, получается, что адрес конечного получателя – хост-машины – состоит из двух чисел: номера PSN-узла и номера порта, к которому подключена эта хост-машина.

Конец 70-ых и начало 80-ых годов XX столетия характеризуются бурным ростом компьютерных сетей, которые были построены по совершенно разным технологиям и предназначены для решения совершенно различных задач. При этом многие понимали, что от возможности обмена информацией между сетями все только бы выиграли. Однако все эти сети использовали совершенно разные протоколы, несовместимые между собой. То есть, возникла необходимость разработки нового протокола, который бы, не нарушая структуры существующих сетей, позволял им обмениваться информацией, и работал бы на любой аппаратной платформе. Среди множества этих сетей особенно выделялась ARPANET. Ее децентрализованная структура, существенно отличающаяся от структур, существовавших в то время корпоративных сетей, позволяла подключать к сети компьютеры практически любого типа, при одном лишь условии – что бы они «понимали» протокол пакетной передачи данных NCP. Но существующий вариант протокола NCP имел ограничения, не позволяющие подключать к ARPANET такое большое количество сетей и компьютеров. В сентябре 1973 года Винт Серф и Боб Кан, входившие в Международную сетевую рабочую группу (INWG), распространили на специальной встрече этой группы, состоявшейся во время конференции в Университете Суссекса, первую документированную версию выработанных ими спецификаций – *семейства протоколов TCP/IP*. В июле 1977 Серф и Кан впервые продемонстрировали передачу данных с использованием TCP по трем различным сетям. Пакет прошел по следующему маршруту: Сан-Франциско – Лондон – Университет Южной Калифорнии. В конце своего путешествия пакет проделал 150 тысяч км, не потеряв при этом ни одного бита! А 1 января 1983 года ARPANET перешла с протокола NCP на TCP/IP. Это был переход в стиле «Дня X», требующий одновременных изменений на всех компьютерах. Переход тщательно планировался всеми заинтересованными сторонами в течение нескольких предшествующих лет и прошел на удивление гладко.

Внедрение протокола TCP/IP, позволявшего пользователям с легкостью подключаться к сети при помощи обычной телефонной линии, совпало с другим событием – разделением ARPANEN. Терпению военных пришел конец – их родная, лелеемая и подкармливаемая серьезными капиталовложениями сеть, перестала обеспечивать необходимый уровень секретности. Поэтому она была разделена на две сети. Первая из них – *Defense Data Network* (*DDN*) – Оборонная Сеть Передачи Данных, известная так же как *MILNET – Military Network* – сеть военных – стала использоваться исключительно для военных целей. Вторая же, ARPANET, по-прежнему использовалась для исследовательских целей. Таким образом, семейство протоколов TCP/IP позволило не только объединять сети, но и разъединять их. Термин «Интернет» стал использоваться для обозначения единой сети: MILNET плюс ARPANET. В 1989 году обслуживание ARPANET полностью прекратилось, и в 1991 году она закончила свое существование. Ее функции продолжила *NSFNET*, объединившая к тому времени большинство сетей и практически ставшая Интернетом в том виде, в котором он сейчас есть.

Официально NSFNET (National Science Foundation Network) была сформирована в 1986 году одним из подразделений Национального Научного Фонда США (NSF – National Science Foundation) – Отделом Сетевых и Коммуникационных Исследований и Инфраструктуры. NSFNET была составлена из более мелких сетей (включая известные тогда сети Usenet и Bitnet) и имела гораздо большую пропускную способность, чем ARPANET. Она основывалась на протоколе TCP/IP, имела иерархическую структуру и концентрировалась вокруг крупных университетских центров США. Чтобы университет мог получить от NSF средства на подключение к Интернету, он, как было записано в программе NSFNET, «должен обеспечить доступность этого подключения для *всех* подготовленных пользователей в университетском городке».

Сеть NSFNET – это пять *суперкомпьютерных центров*, обычно называемые «магистральным хребтом Интернет в США» (*Internet Backbone*),

которые были соединены специальными телефонными линиями с пропускной способностью 56 Кбит/сек. Сеть NSFNET представляла собой крупную опорную сеть, объединяющую вокруг себя более мелкие сети. Любой, кто мог установить связь с сетью NSFNET, получал доступ ко всем объединенным вокруг нее сетям. Она, по сути, была зеркальным отражением ARPANET. Некоторое время они работали параллельно. Региональные сети на базе протокола TCP/IP были соединены друг с другом через NSFNET, которая имела связь с ARPANET. Подключения осуществлялись главным образом через NSFNET, так как она обладала более высокой скоростью передачи данных, к ней легче было подсоединиться и обходилось это дешевле. Название «Интернет» начало плавно переходить от ARPANET к NSFNET.

Совместное использование суперкомпьютерных центров позволяло использовать и множество других вещей, не относящихся к суперкомпьютерам. Неожиданно университеты, школы и другие организации осознали, что заимели под рукой море данных и целый мир пользователей. Поток сообщений в сети (*трафик*) нарастал все быстрее и быстрее пока, в конце концов, не перегрузил управляющие сетью компьютеры и связывающие их телефонные линии. В 1987 году NSF заключил с компаниями Merit Network Inc., IBM и MCI контракт на управление и развитие сети. Последний предполагал замену линий связи, ранее использовавшихся в NSFNET, на магистрали T1 с пропускной способностью 1.544 Мбит/сек, а также объединение шести региональных сетей, пяти существующих суперкомпьютерных центров, сети MERIT и Национального центра атмосферных исследований в единую сеть. Эта работа была завершена в июле 1988 года. Сетевые управляющие машины также были заменены на более совершенные.

Процесс совершенствования и модернизации сети идет непрерывно. Однако большинство этих перестроек происходит незаметно для пользователей. Включив компьютер, вы не увидите объявления о том, что ближайшее время Интернет не будет доступен из-за его модернизации. Возможно, даже более важно то, что изменение сети и ее усовершенствование

создали зрелую и практичную технологию. Проблемы были решены, а идеи развития проверены в деле. Так, в 1991 году опять-таки из-за нехватки пропускной способности NSFNET, ее магистральные каналы были модернизированы до «статуса T3», что соответствует скорости передачи данных 44.738 Мбит/сек.

В 1993 году NSF решил, что не может больше поддерживать быстро растущую систему. Он предложил независимым фирмам контракты на осуществление контроля над расширением Интернет. Компании откликнулись, и ответственность за развитие Интернет была передана множеству различных фирм. Сама же NSFNET была заменена *NAP – Network Access Point* – Точки Доступа Сети, которые были разбросаны по всей территории США. Через них осуществлялось взаимодействие и обмен информацией между компаниями, которые строили свои собственные опорные сети и предоставляли доступ к Интернету конечным пользователям. Количество таких компаний очень быстро росло и они стали называться *ISP – Internet Service Provider* – Провайдер Доступа Интернет, или просто – Провайдер. Первым коммерческим ISP был The World. Контроль и финансирование NAP – высшего уровня иерархии Интернет – остался в NSF. Отдельные провайдеры частных опорных сетей образуют сложную систему связей – *Точек присутствия – Points of Presence – POP*, через которые отдельные пользователи или компании подключаются к Интернет.

NAP начала свою работу в 1995 году, приняв на себя все потоки информации NSFNET. Она имела сверхвысокую пропускную способность – до 155 Мбит/с и состояла из 4-х узлов, расположенных в Вашингтоне, Нью-Йорке, Сан-Франциско и Чикаго. 30 апреля 1995 года опорная сеть NSFNET прекратила функционировать в качестве магистрали – появился Интернет в том виде, который мы знаем сегодня.

С момента создания самого первого варианта ARPANET пользователями сети Интернет были только специалисты в области обработки информации. Однако, пересылка файлов и сообщений электронной почты – это не совсем то,

что нужно большинству рядовых пользователей. Наиболее интенсивное проникновение Интернет буквально во все области человеческой деятельности приходится на 90-е годы XX столетия. И связано это было в первую очередь с создание *Всемирной Паутины – World Wide Web – WWW*. Концепция Всемирной Паутины, в отличие от существующих к тому времени служб Интернет, таких, например, как электронная почта, дала возможность представлять информацию в естественной для человека форме – т.е. с текстом, изображениями, звуком, видео и прочими атрибутами. Документы, выполненные в этом формате, получили название *Web-страниц*. Фактически WWW – это распределенная система, гипертекстовых документов. После этого работа в Интернете перестала быть уделом профессионалов – он превратился в распределенную по миллионам серверов единую информационную базу, навигация в которой не сложнее, чем просмотр обычной мультимедийной энциклопедии.

Создание Всемирной Паутины связано с именем Тима Бернерс-Ли (Tim Berners Lee) – сотрудника Европейского центра ядерных исследований (CERN). Он заложил три, из четырех ныне существующих, краеугольных камней WWW:

1. язык гипертекстовой разметки документов *HTML – HyperText Markup Language*,
2. протокол обмена гипертекстовой информацией *HTTP – HyperText Transfer Protocol*,
3. универсальный способ адресации ресурсов в сети *URL – Universal Resource Locator*,
4. общий интерфейс шлюзов *CGI – Common Gateway Interface*, который был добавлен позже командой из Национального Центра Суперкомпьютерных Приложений США (National Center for Supercomputer Applications – NCSA) .

Первый текстовый *браузер* (*обозреватель* – программа для просмотра Web-страниц), созданный Тимом Бернерс-Ли в 1990 году, особой популярности не имел. Более удачной была версия браузера *Mosaic* для операционной системы UNIX, разработанная в NCSA группой программистов во главе с

Марком Андрисеном (Marc Andreessen) в 1993 года. С 1994, после выхода версий браузера Mosaic для операционных систем *Windows* и *Macintosh*, а вскоре вслед за этим и браузеров *Netscape Navigator* и *Microsoft Internet Explorer*, берет начало взрывообразное распространение популярности WWW, и как следствие Интернета, среди широких масс сначала в США, а затем и по всему миру. А с 1996 года Всемирная Паутина почти полностью подменяет собой понятие «Интернет».

Подводя итог краткой истории создания Интернета, можно выделить три основных этапа ее развития:

1-й этап 1969 – 1983 годы – создание первого варианта сети ARPANET, способной объединять другие сети и различные компьютеры;

2-й этап 1983 – 1990 г.г. – замена протокола NCP на TCP/IP и создание сети NSFNET;

3-й этап 1990 г. – создание Всемирной паутины.

Основные этапы и события истории создания Интернет представлены в таблице 5.1.

Таблица 5.1

Основные этапы создания Интернет

Этап	Год	Событие
I	1969	Создан первоначальный вариант сети ARPANET, состоящий из 4-х компьютеров.
	1970	Сеть ARPANET объединяет 10 компьютеров. Создана первая версия протокола NCP.
	1971	Сеть ARPANET объединяет 15 компьютеров.
	1972	Создано первое «сетевое» приложение – электронная почта. Демонстрация ARPANET на Международной конференции по компьютерным коммуникациям.
	1973	Впервые к ARPANET подключены зарубежные узлы. Выработана 1-я документированная версия семейства протоколов TCP/IP.
	1975	ARPANET превратилась из экспериментальной сети в рабочую.
	1977	Демонстрация передачи данных с использованием TCP/IP по трем различным сетям.

II	1983	Переход ARPANET с протокола NCP на TCP/IP. Из ARPANET выделилась MILNET. Появился термин Интернет.
	1984	Введена система доменных имен – DNS – Domain Name System.
	1986	Создана NSFNET.
III	1990	Создана Всемирная паутина – WWW – World Wide Web.
	1991	ARPANET прекратила свое существование.
	1995	Создана NAP. Закрыта опорная сеть NSFNET.

24 октября 1995 года Федеральный Сетевой Совет (FNC) единодушно одобрил резолюцию, определяющую термин «Интернет». Это определение разрабатывалось при участии специалистов в области сетей и в области прав на интеллектуальную собственность.

РЕЗОЛЮЦИЯ: Федеральный Сетевой Совет признает, что следующие словосочетания отражают наше определение термина «Интернет». Интернет – это глобальная информационная система, которая:

1. логически взаимосвязана пространством глобальных уникальных адресов, основанных на Интернет-протоколе (IP) или на последующих расширениях или преемниках IP;
2. способна поддерживать коммуникации с использованием семейства Протокола управления передачей/Интернет-протокола (TCP/IP) или его последующих расширений/преемников и/или других IP-совместимых протоколов;
3. обеспечивает, использует или делает доступной, на общественной или частной основе, высокоуровневые сервисы, надстроенные над описанной здесь коммуникационной и иной связанной с ней инфраструктурой.

Винтон Серф, президент Сообщества Интернета, дает более краткое определение Интернета так: «Интернет – это глобальная сеть сетей, взаимосвязанных протоколами TCP/IP и другими коммуникационными протоколами».

2. Структура и принципы работы Интернета

Интернет основана на идее существования множества независимых сетей почти произвольной архитектуры. Сам же термин «Интернет» означает – «Междусетье», сообщество сетей разного масштаба. То есть, единой сети фактически нет, а существует множество отдельных сетей, самого различного назначения и структуры, объединенных в единое целое. Одни из таких сетей могут объединять компьютеры в Вашем доме, а другие – принадлежать крупным корпорациям и соединять компьютеры, расположенные на разных континентах. Отдельные сети в составе Интернет относительно независимы и могут развиваться по своим собственным законам и правилам, оставаясь в то же время частью единой структуры. Интернет не является единым целым и никому не принадлежит, но при этом более мелкие сети, подключенные к ней, обслуживаются отдельными организациями - *провайдерами*, являющимися собственниками «своего» участка Сети и получающими плату за предоставление доступа к ней.

Сейчас в Интернет используются практически все известные линии связи от низкоскоростных телефонных линий до высокоскоростных цифровых спутниковых каналов. Операционные системы, используемые в Интернет, также отличаются разнообразием. Большинство компьютеров сети работают под управлением операционной системы Unix или Microsoft Windows.

Основное, что отличает Интернет от других сетей – это ее протоколы – TCP/IP. Свое название протокол TCP/IP получил от двух протоколов *TCP – Transmission Control Protocol – Протокол Управления Передачей* и *IP – Internet Protocol – Межсетевой Протокол*. Первый из них регламентирует способы передачи информации, а второй – способы адресации. Несмотря на то, что в Интернет используется большое число других протоколов, ее часто называют TCP/IP-сетью, так как эти два протокола, безусловно, являются важнейшими.

Когда мы пытаемся представить себе, что же такое Интернет и как она работает, то вполне естественно у большинства из нас возникают ассоциации с телефонной сетью. В конце концов, обе эти структуры используют электронные

средства передачи, обе позволяют устанавливать соединение и передавать информацию; кроме того, в Интернет очень часто используются телефонные линии. К сожалению, это неверное представление, и оно является причиной непонимания принципов работы Интернет. Телефонная сеть – это *сеть с коммутацией каналов*. Когда производится вызов, абоненту выделяется некоторая часть этй сети. Даже когда абонент не используете ее (например, пошел позвать другого человека), она остается недоступной для других абонентов, которым в этот момент нужно позвонить. Это приводит к тому, что такой дорогой ресурс, как сеть, используется неэффективно.

Более соответствующая действительному положению вещей модель Интернета – это почтовое ведомство. Оно представляет собой *сеть с коммутацией пакетов*. Здесь нет выделенного участка сети. Почтовое отделение не будет бронировать самолет, чтобы доставить письмо по назначению – оно смешивается с другими письмами, сортируется и лишь затем – пересылается. Несмотря на то, что технологии абсолютно разные, служба доставки почты представляет собой удивительно точный аналог сети Интернет. Этой моделью будем продолжать пользоваться и далее.

Например, для того, что бы письмо дошло до адресата его необходимо вложить в конверт, написать на конверте адрес и бросить в почтовый ящик. Далее оно поступает в почтовое отделение, сортируется и вместе с другой корреспонденцией при помощи различных транспортных средств, таких как автомобили, самолеты, поезда и т.д., доставляется в другое почтовое отделение, более близкое к месту назначения. В нем письмо подвергается такой же процедуре с целью перемещения к еще более близкому, к месту назначения, почтовому отделению. И так далее, до тех пор, пока, в конце концов, письмо не попадет в нужное почтовое отделение, откуда оно будет доставлено адресату. При этом все почтовые отделения располагают информацией:

1. о своих связях с другими почтовыми отделениями, и
2. о том, какой из маршрутов будет наилучшим для перемещения корреспонденции ближе к пункту назначения.

Аналогично почтовому ведомству работает и компьютерная сеть Интернет. Только вместо писем в ней перемещаются блоки информации, которые называются *пакетами*. Каждый такой пакет помещается в специальный «конверт» – *IP-адрес*, который соответствует нужному компьютеру в сети. Аналогами почтовых отделений являются *маршрутизаторы*. Они способные по IP-адресу принятого пакета автоматически определить, на какой из соседних маршрутизаторов его необходимо переправить. Маршрутизатором может быть программа, но может быть и отдельный, специально выделенный для этой цели компьютер. Наиболее широко в Интернет используются маршрутизаторы типа NetBlazer или Cisco, чья операционная система напоминает Unix. Каждый из них непрерывно общается со смежными маршрутизаторами и потому знает состояние своего окружения. Он знает, когда какой-то из соседей «закрыт» на техническое обслуживание или просто перегружен. Принимая решение о переправке проходящего пакета, маршрутизатор учитывает состояние своих соседей и динамически перераспределяет потоки так, чтобы пакет ушел в том направлении, которое в данный момент наиболее оптимально.

Поскольку к Интернет могут подключаться сети, работающие на основе своих, не TCP/IP, а других, протоколов, то для обмена информацией с ними используются специальные маршрутизаторы – *шлюзы* (*gateways*). Опять-таки, шлюзом может быть специальный компьютер, но это может быть и специальная программа. Шлюзы выполняют преобразование данных из форматов, принятых в локальной сети, в формат, принятый в Интернете, и наоборот.

Аналогами транспортных средств (автомобилей, самолетов, поездов и т. д.), используемых в почтовых ведомствах, в Интернет являются каналы связи. Так же как и транспортные средства, каналы связи могут быть самыми различными. Они могут быть собственными или арендованными, быстрыми (цифровые спутниковые) или медленные (линии местных телефонных компаний), надежными и ненадежными и т. д.

Таким образом, в сети Интернет доставка пакетов данных от компьютера-источника до компьютера-получателя выполняется по различным каналам связи через серию маршрутизаторов на основе Межсетевого Протокола (Internet Protocol) IP, который отвечает за адресацию компьютеров в сети. Но даже хорошо разработанная система адресации не решает полностью всех проблем пересылки информации в сети.

Аналогично почтовому ведомству, при пересылке информации в Интернете возникают и такие проблемы.

- Во-первых, рассмотрим следующий пример. Как следовало бы поступить в случае, если необходимо послать кому-нибудь книгу, а почта принимает только письма? Выход один: вырвать из книги все страницы, вложить каждую в отдельный конверт и бросить все конверты в почтовый ящик. Получателю пришлось бы собирать все страницы (при условии, что ни одно письмо не пропало) и склеивать обратно в книгу. В сети Интернет, аналогично рассмотренному выше примеру, по целому ряду технических причин (в основном это аппаратные ограничения) информация так же разбивается на порции, называемые пакетами. В одном пакете обычно пересылается от одного до 1500 байт информации. Это не дает возможности одному пользователю монополизировать сеть, однако позволяет каждому рассчитывать на своевременное обслуживание. Это также означает, что в случае перегрузки сети качество ее работы несколько ухудшается для всех пользователей, но она не перестанет работать полностью, даже если некоторый ее участок будут монополизирован несколькими солидными клиентами.
- Во-вторых, может произойти ошибка. Почтовое ведомство иногда теряет письма, а сети иногда теряют пакеты или повреждают их при передаче. В Интернет, в отличие от почтового ведомства, эта проблема успешно решается.
- В-третьих, последовательность доставки пакетов может быть нарушена. Например, если по одному адресу одно за другим были посланы два письма,

то нет никакой гарантии, что они пойдут по одному маршруту или придут в порядке их отправления. Такая же проблема существует и в Internet.

Для решения всех этих задач и предназначен следующий уровень протоколов – Протокол Управления Передачей (Transmission Control Protocol) TCP, который является второй, неотъемлемой, частью семейства протоколов TCP/IP. То есть, протокол TCP отвечает за:

1. разбивку сообщения на отдельные пакеты,
2. обработку ошибок передачи пакетов, и
3. восстановления необходимой последовательности пакетов на принимающей стороне.

3. Адресация компьютеров в сети

Каждый компьютер, подключенный к Интернет, обязательно имеет свой уникальный адрес, называемый *IP-адресом* (читается «ай-пи»). IP-адрес компьютера может быть постоянным, или каждый раз назначаться новым при соединении с сетью. Но всегда один IP-адрес соответствует только одному компьютеру.

Длина IP-адреса – 4 байта (по сетевой терминологии *октета*) или 32 бита. Для удобства эти адреса записываются в виде четырех чисел от 0 до 255, разделенных точкой.

Например:

204.146.46.133 – IP-адрес сервера компании Microsoft,

207.68.137.53 – IP-адрес сервера компании IBM.

Теоретически к сети одновременно могут быть подключено до $2^{32} = 4\ 294\ 967\ 296$ компьютера. На самом деле, некоторые комбинации битов зарезервированы и это число намного меньше. Так, например, адрес 127.0.0.1 используется только на локальном компьютере.

Любой IP-адрес состоит из 2-х частей:

1. *адреса сети* и

2. *адреса хоста* (*хостом* называют любой, подключенные к Интернет, компьютер).

В зависимости от того, какая часть IP-адреса выделена под адрес сети, а какая – под адрес компьютера в ней, различают сети классов *A*, *B* и *C* (существуют также служебные классы *D* и *E*). Сети класса *A* – это огромные сети, но их не много. И наоборот, сетей класса *C* очень много, но компьютеров у них мало.

Чтобы отделить адрес сети от адреса хоста, используется *маска подсети*, также представляющая собой 32-битное число. По умолчанию сетям класса *A* соответствует маска 255.0.0.0, класса *B* – 255.255.0.0, а сетям класса *C* – 255.255.255.0. То есть, в двоичном представлении маски, позиции, соответствующие адресу сети, закрыты единицами. Маска подсети может использоваться и для других целей, например, для логического разделения локальных сетей на подсети меньшего масштаба.

Пользователю Интернет совсем не обязательно вникать во все подробности структуры IP-адреса – он устанавливается по соглашению между соседними маршрутизаторами и имеет значение только при создании сети.

4. Доменная система имен

Хост-файл

Создание DNS

Пространство доменных имен

Механизм разрешения

Регистрация доменов

dotCOM-бизнес

Хост-файл

Цифровой IP-адрес хоста удобен для его автоматической обработки компьютером, но крайне не удобен для восприятия человеком – люди предпочитают более-менее осмысленные имена. Поэтому, с самого начала существования Интернета, компьютеры помимо цифровых IP-адресов могли

иметь и символьные имена. Вначале это был так называемый *хост-файл*. Хост-файл – это обыкновенный текстовый файл, который так и назывался *hosts.txt*. Он представлял собой таблицу, каждая строка которой состояла из пары «*имя хоста – его IP-адрес*». Файл hosts.txt похож на телефонный справочник, где каждому номеру телефона соответствует имя владельца. Если необходимо позвонить какому-либо человеку, то по телефонной книге всегда можно найдем его номер (при условии, что он был заранее зарегистрирован). Авторство создания этого файла-таблицы принадлежит Джону Постелу (Jon Postel), который первым его и поддерживал. Для присвоения имени хосту необходимо было обратиться в специальную службу регистрации *IANA Сетевого Информационного центра* (*Network Information Center – NIC*) в *Стенфордском Исследовательском Институте* (*Stanford Research Institute – SRI*, Menlo Park, California) которая:

1. добавляла в файл hosts.txt запись с неповторяющимся именем хоста и его IP-адрес, а также
2. регулярно рассылала этот файл на все остальные компьютеры сети.

В качестве имен хостов использовались простые английские слова, каждое из которых обязательно являлось уникальным. Таким образом, поддерживалась актуальность файла hosts.txt и, соответственно, имен компьютеров и их IP-адресов в сети.

⚠ *IANA – Internet Assigned Numbers Authority принято переводить на русский как «Уполномоченная организация по распределению нумерации в сети Интернет». На самом деле, IANA – это не организация, а набор функций. То есть IANA сама по себе не имеет организационно-правовой формы. В настоящее время исполнение IANA-функций возложено на корпорацию* ***ICANN*** *–* ***The Internet Corporation for Assigned Names and Numbers*** *–* ***Интернет-корпорация по распределению адресов и имен****. К основным функциям IANA относятся:*

- *координация работ по выработке технических параметров протоколов;*

- *административные функции по управлению корнем системы доменных имен;*
- *распределение блоков IP-адресов;*
- *прочие функции, которые могут быть включены в IANA-функции по согласованию с Министерством торговли США.*

Создание DNS

Однако, как показало время, такой подход оказался тупиковым, поскольку поддерживать актуальность хост-файла в связи с бурным ростом Интернета и, соответственно, числа компьютеров в ней, становилось все труднее и труднее. Решение этой проблемы было найдено в 1983 г., когда Пол Мокапетрис (Paul Mockapetris) из Института информатики Университета Южной Калифорнии (USC/ISI) в *RFC-882* и *RFC-883* формально описал *Доменную Систему Имен* (*Domain Name System – DNS*), которая должна была заменить изживший себя хост-файл. То есть DNS решала проблему, с которой не справилась хост-таблица, используя две концепции:

1. иерархию имен, и
2. распределение ответственности.

⚠ *RFC – Request for Comments – документы с таким названием содержат в себе материалы по Интернет-технологиям, которые доведены до уровня стандарта или близки к этому уровню.*

⚠ *Работа по созданию DNS была начата в 1984 г. четырьмя старшекурсниками университета в Беркли: Дугласом Терри (Douglas Terry), Марком Пойнтером (Mark Painter), Дэвидом Ригглом (David Riggle) и Сонг Ньян Чжоу (Songnian Zhou). Эстафету подхватил Ральф Кэмпбелл (Ralph Campbell) из Computer Systems Research Group, который начал «склеивать» доменную систему имен в BSD-UNIX. В 1985 г. Кевин Данлэп (Kevin Dunlap), инженер DEC, временно работавший в Беркли, принял этот проект в свои руки и создал систему BIND (Berkeley Internet Name Domain – систему доменных имен Интернета реализации Беркли). Майк Кареле (Mike Karels) и Пол Викси (Paul Vixie) сопровождали эту систему в течение ряда лет. Пол*

продолжает вести ее и сейчас, пользуясь помощью участников телеконференции isc.org и членов списка рассылки bind-workers.

DNS – это, прежде всего, один из ключевых элементов инфраструктуры Интернета, который обеспечивает трансляцию алфавитно-цифровых имен компьютеров в их цифровые IP-адреса, необходимые для физического доступа к ним. Например, символьному адресу www.yandex.ru/ соответствует IP-адрес 213.180.194.129, по которому, собственно, и осуществляется доступ к хосту. Фактически DNS выполняет функции «человеко-машинного интерфейса» между пользователями Интернета с одной стороны и системами адресации и маршрутизации с другой стороны.

Доменная система имен, как один из множества сервисов (служб) Интернета, выполняет две основных функции:

1. организацию *пространства доменных имен*, и
2. обеспечение *механизма разрешения*, т.е. сопоставление символьного имени компьютера его цифровому IP-адресу.

Пространство доменных имен

Пространство доменных имен (*Domain Name Space*) сменило плоскую систему именования компьютеров в сети, реализованную при помощи хост-файла, на иерархическую структуру, допускающую наличие в имени произвольного количества составных частей. Фрагмент такой структуры представлен на рисунке 5.2.

Пространство доменных имен аналогично иерархии имен файлов, принятой в большинстве файловых систем. Оно *начинается с корня, который имени не имеет* и называется *корневым доменом* или *доменом нулевого уровня*. Затем следуют домены *первого уровня*, *второго уровня* и т.д. уровней.

Последний уровень пространства доменных имен (лист) соответствует конечному узлу сети (хосту) – реальному компьютеру с соответствующим IP-адресом. В отличие от имен файлов, при записи которых сначала указывается самая старшая составляющая, затем составляющая более низкого уровня и т.д., запись доменного имени начинается с самой младшей составляющей, а

заканчивается – самой старшей. Составные части доменного имени друг о друга отделяются точкой. Например, в имени www.ksame.kharkov.ua составляющая www является именем одного из компьютеров в домене ksame.kharkov.ua.

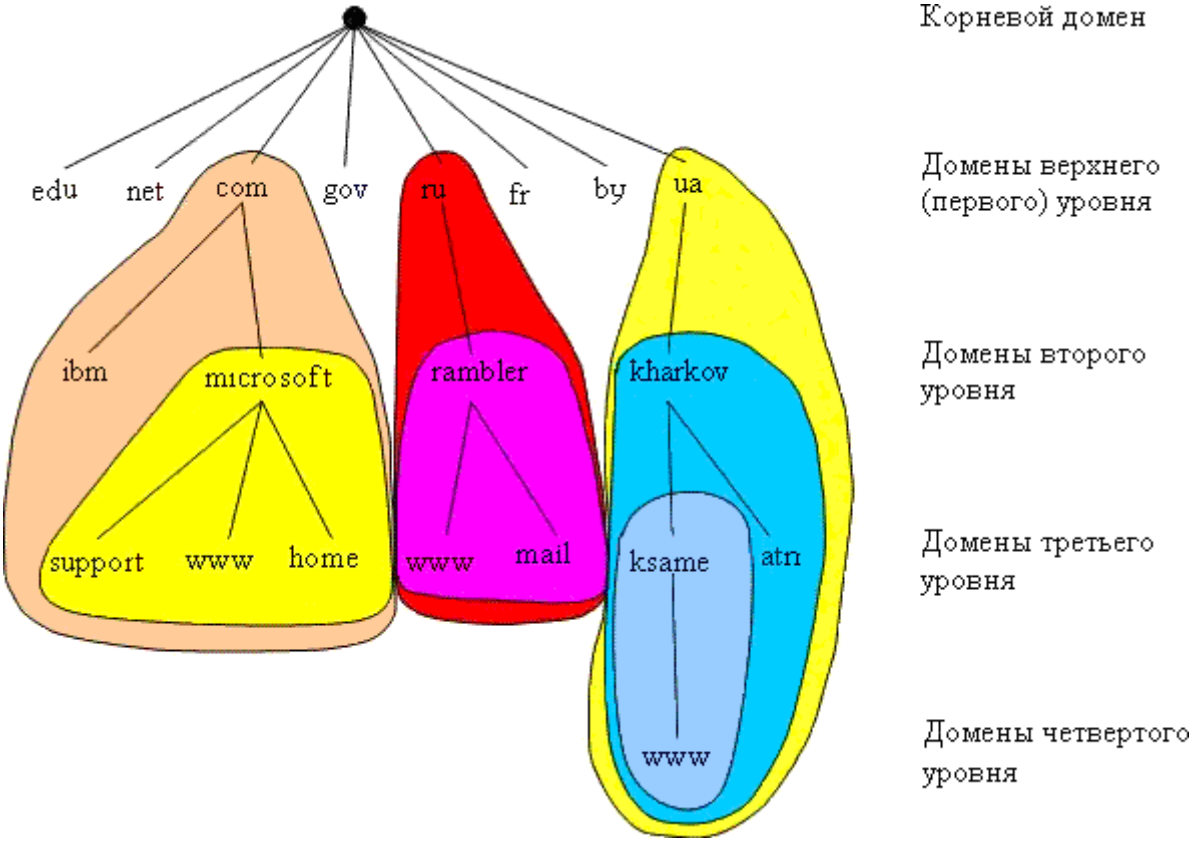


Рисунок 5.2 – Фрагмент пространства доменных имен

Имена компьютеров бывают двух видов:

1. собственные и
2. функциональные.

Собственное имя компьютера назначаются его владельцем или администратором домена произвольно. Функциональные имена вытекают из функций, которые он выполняет. Например:

www – сервер Всемирной Паутины (WWW),

ftp – FTP-сервер,

mail – почтовый сервер и т.д.

Домены первого уровня (*Top Level Domains – TLDs*) делятся на две категории:

1. домены общего назначения (*Generic TLDs – gTLDs*) и

2. национальные домены (*Country Code TLDs – ccTLDs*).

Домены общего назначения (организационные, родовые – в зависимости о то рода деятельности) – это, как правило, трехбуквенные имена, предназначены для использования всем Интернет-сообществом. Наиболее широко используемыми (наиболее «старыми») доменами общего назначения являются следующие:

Название	Сокращение от	Род деятельности (использование)
com	**com**mercial	Коммерческие
edu	**edu**cational	Образовательные
gov	**gov**ernment	Правительственные
mil	**mil**itary	Военные
net	**net**work	Обеспечивающие работу сети
org	**org**anization	Некоммерческие

Изначально gTLDs предназначались для объединения доменов нижних уровней, принадлежащих организациям и учреждениям США. Однако ничто не мешает какой-нибудь компании, например, российской или украинской, зарегистрировать свой домен второго уровня в домене «com». Исключением являются только домены «mil» и «gov», которые используются только учреждениями и организациями США.

Национальные (региональные, географические) домены – это двухбуквенные имена, которые обозначают коды стран в соответствии со стандартом ISO 3166-1. Например, «ua» – Украина, «ru» – Россия, «by» – Беларусь, «fr» – Франция и т.д. Для США наименование страны по традиции опускается – поскольку Интернет зародилась и развивалась в США – там самыми крупными объединениями являются домены общего назначения (gTLDs). Однако, для обеспечения единообразия, США имеют свой национальный домен – «us». Для бывшего СССР также был выделен домен – «su», который поддерживается, но новые домены более низких уровней в нем больше не регистрируются.

Каждый домен верхнего уровня включает в себя домены (*поддомены*) второго уровня, имена которых выбираются относительно произвольно, например, по имени компании, за которой зарегистрировано это имя, или по

названию региона. Порядок создания доменов второго уровня определяется администратором соответствующего *родительского* домена первого уровня.

В доменах (зонах) государств, опять же, могут быть и организационные и географические зоны. Организационные домены в большинстве своем повторяют структуру организационных доменов верхнего уровня. Географические зоны выделяются по городам, областям и другим территориальным образованиям. Непосредственно и в тех и в других размещаются домены третьего уровня, обычно относящиеся к организациям, подразделениям внутри компаний или домены персональных пользователей.

Общие правила построения доменных имен следующие:

- имя может состоять только из букв латинского алфавита, цифр и символа «-» (дефис);
- длина каждой составляющей доменного имени не может превышать 63 символов;
- общая длина доменного имени не может превышать 255 символов;
- отдельные составляющие доменного имени друг от друга отделяются символом точка «.»;
- доменные имена являются нечувствительными к регистру символов, входящих в его состав, например, последовательности символов «Com», «COM», «cOm», «com» и т.п. обозначают одно и то же имя; реализации DNS должны игнорировать регистр символов при сравнениях, но обязаны распространять его, если он указан.

Разделение доменного имени на части позволяет разделить административную ответственность за назначение уникальных имен в пределах своей компетенции. Так, например, администратор домена «ua» несет ответственность за то, чтобы все имена в его зоне ответственности были неповторяющимися. Это позволяет решить проблему образования уникальных имен без взаимных консультаций – если в каждом домене обеспечивается уникальность имен следующего уровня иерархии, то и вся система доменных имен будет состоять из уникальных имен. Таким образом, каждый компьютер

(узел, хост) в сети Интернет однозначно определяется своим *полным доменным именем* (*Fully Qualified Domain Name – FQDN*), которое включает имя хоста и имена всех доменов по направлению от узла (листа) к корню дерева. Например, (рисунке 5.2) компьютер имеет имя «mail» (почтовый сервер). Если собрать имена всех доменов по структуре от листа до корня дерева, то получится полное доменное имя – «mail.rambler.ru». По аналогии с файловой системой в пространстве доменных имен различают также *краткие* и *относительные имена*. Краткое имя – это имя конечного узла сети (хоста, листа), например, (рисунке 5.2) «atn» в домене «kharkov.ua». Относительное имя – это составное имя, начинающееся с некоторого уровня иерархии, но не с самого верха. Например, (рисунке 5.2) «www.ksame» в домене «kharkov.ua».

Таким образом, доменное пространство имен – это метод назначения символических имен хостам путем передачи ответственности сетевым группам за их подмножество имен. В результате адрес того или иного ресурса Интернет, записанный в стандарте DNS, дробится на несколько составляющих, отделенных друг от друга точкой. Каждый из этих элементов, кроме самого левого, носит название *домен* (англ. *domain* – область, сфера, зона и т.д.). То есть, домен представляет собой некое логическое подмножество сетевых ресурсов, имеющее собственное имя и управляемое из единого центра.

 *Следует иметь в виду, что доменные имена в реальной жизни достаточно причудливо отображаются на IP-адреса реальных компьютеров, которые физически подключены к сети. Так, например, компьютер, физически установленный и подключенный к Интернету в далекой Америке, может совершенно спокойно иметь имя из украинского корпоративного домена, например, www.salo.ua, и наоборот, компьютер украинского сегмента может иметь имя из домена «com». Последнее, к слову сказать, встречается гораздо чаще.*

Более того, один и тот же компьютер может иметь несколько доменных имен; и наоборот – возможен вариант, когда за одним доменным именем может быть закреплено несколько IP-адресов, которые реально

*назначены различным компьютерам. Таким образом, соответствие между доменными именами и IP-адресами в рамках системы доменных имен не является взаимно однозначным, а строится по схеме «**многие-ко-многим**».*

Доменное пространство имен, возможно, и выглядит несколько сложновато, но это одна из составляющих, которая делает общение с сетью более простым и удобным. Несомненное преимущество доменного пространства имен состоит в том, что оно разбивает Интернет на набор вполне обозримых и управляемых частей. Хотя сеть включает миллионы компьютеров, все они поименованы, и именование это организовано в удобной и рациональной форме.

Механизм разрешения

Механизм разрешения, т.е. сопоставление символьного имени компьютера его цифровому IP-адресу в сети, построен по схеме «клиент-сервер» и состоит из двух частей:

1. DNS-клиента (*name-resolver*) – программы, выполняющейся на компьютере клиента, которой требуется найти по доменному имени IP-адрес компьютера, и
2. DNS-сервера (*name-server*) – программы, осуществляющей по запросу DNS-клиента поиск IP-адреса на основе предложенного ей доменного имени.

Параметры программного обеспечение DNS-клиента настраиваются во время подключения компьютера к сети. Так, при подключении компьютера к локальной сети в параметрах сетевого подключения стека протоколов TCP/IP необходимо указать IP-адрес первичного (предпочитаемого) локального сервера имен, а также, возможно, вторичного (альтернативного) DNS-сервера. При непосредственном же подключении компьютера к Интернету, адреса первичного и вторичногоDNS-серверов могут быть настроены как вручную, так и автоматически. То есть, в последнем случае локальным сервером имен буден DNS-сервер Интернет-провайдера.

DNS сервера – это компьютеры, на которых установлено специальное программное обеспечение. Каждый DNS-сервер обслуживает свою *зону ответственности* в иерархическом пространстве доменных имен.

⚠ *Часть иерархического пространства имен DNS, обслуживаемая сервером имен и представленная в его локальной базе данных, называется* ***зоной ответственности (Zone Of Authority)***.

Важным преимуществом системы доменных имен является то, что каждый конкретный DNS-сервер имен не должен содержать в своей локальной базе данных описание всей иерархии пространства доменных имен. В его базе данных представлено только пространство имен, принадлежащих его домену. Например, (рисунке 5.2) если в домене «kharkov.ua» имеется только два узла – «ksame» и «atn», то в базе данных сервера имен этого домена может присутствовать всего две записи для указанных имен. Кроме того, если домен содержит в себе домены нижних уровней, то каждый такой поддомен (*субдомен*) может иметь свой собственный сервер имен, освобождая тем самым от необходимости обслуживать свое подпространство имен DNS-сервер родительского домена. Например, (рисунке 5.2) поддомен «ksame» в домене «kharkov.ua». Таким образом, продолжая пример (рисунке 5.2), сервер имен домена «ua» не имеет в своей базе данных информации об узлах «ksame» и «atn» домена «kharkov.ua». Такая передача полномочий по управлению именами части зоны ответственности называется *делегированием*.

Таким образом, DNS-сервера представляют собой распределенную иерархическую базу данных, которая содержит информацию обо всех компьютерах (хостах), подключенных к сети Интернет. Для нормального функционирования системы доменных имен каждому DNS-серверу не обязательно знать адреса всех остальных DNS-серверов Интернет. Для этого ему достаточно располагать IP-адресами:

1. корневых DNS-серверов, а также
2. всех DNS-серверов делегированных зон (поддоменов, субдоменов).

Для распределения нагрузки, а также повышения отказоустойчивости всей системы доменных имен в каждой зоне ответственности одновременно и параллельно функционирует несколько DNS-серверов. То есть DNS-сервера не существуют в единственном экземпляре. У каждого из них (еще называют первичным, основным, предпочитаемым DNS-сервером) есть один или несколько «братьев-близнецов» (вторичных, альтернативных DNS-серверов). Вся информация, записанная на первичном DNS-сервере, периодически копируется на все остальные. Это дает возможность нормально работать всей системе DNS в случае выхода из строя любого из серверов. Так, за поддержку самого верхнего, корневого, домена *на сегодняшний день* отвечают тринадцать *корневых DNS-серверов* (*Root Servers*). Десять из них расположены в США, два – в Европе и один – в Японии. На них, прежде всего, содержится информация о серверах, которые поддерживают домены первого уровня.

Алгоритм работы службы DNS достаточно прост. Когда программе-клиенту требуется по доменному имени выяснить IP-адрес, она связывается с локальным DNS-сервером имен, адрес которого указан в настройках сетевого протокола TCP/IP. Локальный сервер имен, получив такой запрос, рассматривает его, чтобы выяснить, в каком домене находится указанное имя. Если указанный домен входит в его зону ответственности, то сервер преобразует имя в IP-адрес на основе собственной базы данных и возвращает результат клиенту. Если же запрашиваемое доменное имя не входит в его базу данных, то он переадресует запрос вышестоящему DNS-серверу. В любом случае по указанному доменному имени программа-клиент всегда получит необходимый ей IP-адрес.

Таким образом, доменная система имен позволила создать масштабируемый распределенный механизм для отображения иерархических имен компьютеров в их цифровые IP-адреса.

Регистрация доменов

Зарегистрировать доменное имя можно двумя способами:

1. самостоятельно, изучив инструкции на соответствующих серверах, или

2. обратиться к поставщику услуг Интернет, который возьмет на себя все хлопоты по регистрации доменного имени.

В последнем случае главное необходимо проследить, чтобы домен был зарегистрирован именно на Вас или Вашу компанию, а не на Интернет-провайдера. Обработка заявки на регистрацию домена занимает около двух недель.

Процессом распределения IP-адресов и доменных имен *первого уровня* управляет неправительственная, некоммерческая организация ICANN (The Internet Corporation for Assigned Names and Numbers – Интернет-корпорация по распределению адресов и имен) имеющая адрес www.icann.org. Полномочия по распределению доменных имен в зонах *второго уровня* ICANN передает другим уполномоченным организациям – *регистраторами*. Регистраторы, в свою очередь, могут предоставлять другим организациям возможность регистрировать домены через свои партнерские программы. Регистрацию доменов второго уровня на территории США осуществляет InterNIC (Internet Network Information Center), которая находится по адресу www.internic.net. В Европе ее функцию взяла на себя организация RIPE (фр. Réseaux IP Européens), имеющая адрес www.ripe.net. В России регистрацией доменов в зоне «ru» занимается RIPN с адресом www.ripn.net.

Найти другого регистратора в каком-либо конкретном региональном или международном домене тоже не сложно. В региональных доменах, обычно, есть сайты с адресами вида «www.nic.*domain*». То есть, для украинского домена это будет «www.nic.ua», для российского – «www.nic.ru», а для белорусского – «www.nic.by». Если же подходящего сайта не оказалось, можно просто выполнить поиск вида «регистрация доменов ua» и сразу получить список многих регистраторов украинского домена (запрос желательно писать на соответствующем языке).

Регистрация домена, чаще всего, платная, т.к. и корневые сервера и сервера, обслуживающие региональные домены, требуют технической поддержки. То есть домен покупается. Тем самым приобретается право

владения доменом на какой-то определенный срок (минимум 1 год), подлежащий продлению (также платному). Владение доменом подразумевает, что его владелец имеет право полностью распоряжаться тем, какие сервера будут обрабатывать конечные запросы в этом домене, т.е. будут *авторитетными* для данного домена.

Организации или физическому лицу, желающим зарегистрировать свой домен, следует обращаться к администратору какого-либо уже существующего домена. Однако в любом случае предварительно необходимо проверить, не зарегистрировано ли уже такое имя. Это можно сделать следующим образом. В большинстве зон ответственности имеется свой сервер «whois», через который можно узнать подробную информацию о человеке или организации, на которую зарегистрирован тот или иной домен. Например, в зоне «ua» имеется сервер whois.com.ua. Он также отвечает за домены второго уровня, такие как «net.ua», «com.ua» и т.д. Для зоны «ru» информацию о доменном имени можно получить также по адресу www.ripn.net/nic/whois/, а для доменов «com», «org», «net», «edu» и т.д. – по адресам www.register.com или www.internic.net/whois.html. Если же выбранное имя уже зарегистрировано, то остается попытаться придумать другое. Также можно попробовать выйти на организацию или частное лицо, владеющее данным доменом, и попытаться его перекупить.

dotCOM-бизнес

С точки зрения технологии сетей TCP/IP, на которых строится межсетевое взаимодействие в Интернете, DNS является вспомогательной прикладной службой. Однако значение DNS для нужд навигации в Интернете трудно переоценить. Оно выходит далеко за пределы простого преобразования «имя–адрес».

Целая отрасль экономики *dotCOM* получила название от доменного имени «.com». Таким образом, люди, в руках которых находится управление системой DNS или отдельными ее компонентами, обладают «магической» властью. По их неосторожности или злому умыслу из Интернета могут «исчезнуть» не

только отдельные компании или персоны, но и целые страны, как это произошло, например, 9 апреля 2004 года с доменным именем Ливии.

Доменное имя в новых условиях стало выполнять функции бренда, торговой марки, наименования товара и т.д. Это, в свою очередь, привело к борьбе за право управления соответствующими доменами; возникла необходимость создания законодательной базы для решения таких споров. Взрывной рост количества коммерческих информационных ресурсов в Интернете в период с 1994 по 1998 год позволил перевести поддержку инфраструктуры Интернета на самоокупаемость. Управление серверами, реестрами и регистрация доменных имен стали самостоятельными видами бизнеса.

5. Единообразный локатор ресурса

Единообразный локатор (определитель местонахождения) ресурса (*URL – Uniform Resource Locator*), который ранее назывался *Universal Resource Locator* – универсальный локатор ресурса – *это стандартизированный способ записи адреса ресурса в сети Интернет*. URL был разработан Тимом Бернерсом-Ли в 1990 году, как одна из составляющих Всемирной паутины (WWW). Изначально он предназначался для максимально естественного указания мест расположения во Всемирной паутине таких ресурсов, как Web-страницы.

URL состоит из двух частей:

1. названия схемы доступа к ресурсу (обычно – название протокола) и
2. специфической для данной схемы информации, например, название и местоположения ресурса, а также метод доступа.

В целом же синтаксис URL представляет собой расширение принятого на локальных компьютерах понятия *полного имени файла* применительно к множеству ресурсов Интернета и выглядит следующим образом:

[<**протокол**>://][<**логин**>:<**пароль**>@][<**хост**>][:<**порт**>][/<**путь**>][<**имя файла**>][<**дополнительная информация**>]

Компоненты в этой записи является необязательным (все они помещены в квадратные скобки []) и выполняют такие функции:

протокол – сетевой протокол (название схемы доступа) для обращения к ресурсу является первой частью URL и определяет его вторую, специфическую, часть. Наиболее распространенными протоколами в сети Интернет являются следующие:

http – протокол передачи гипертекста,

https – специальная реализация протокола HTTP, использующая шифрование (как правило, SSL или TLS),

mailto – адрес электронной почты,

ftp – протокол передачи файлов,

telnet – интерактивная сессия Telnet,

file – имя файла на локальном компьютере,

data – непосредственные данные.

От остальной части URL протокол отделяется двоеточием (:) и двумя наклонными линиями – двойным слешем (//).

логин – имя пользователя, используемое для доступа к ресурсу.

пароль – пароль, соответствующий указанному имени пользователя.

Логин от пароля отделяется символом «двоеточие» (:). Они используются не со всеми протоколами. Например, с протоколом telnet они применяются для «входа» практически в любой компьютер Интернета, а с протоколом HTML – никогда, поскольку могут быть использованы злоумышленниками. Если же логин и пароль присутствуют, то от оставшейся части URL они отделяются символом «собачка» (@).

хост – доменное имя хоста или его IP-адрес.

порт – это некоторое условное число в диапазоне от 1 до $2^{16} - 1 = 65535$, позволяющие различным программам, выполняющимся на одном компьютере, получать данные независимо друг от друга. Каждая программа обрабатывает данные, поступающие на определенный порт (иногда говорят, что программа

«слушает» этот номер порта). Как правило, в URL номер порта явно не указывается – используется значение по умолчанию, например, значение 80 для протокола HTTP, 23 – для telnet, 21 – для команд, а 20 – для данных в протоколе FTP и т.д.

путь – задает папку (каталог) на сервере, в которой содержится необходимый ресурс (как правило, файл).

⚠ *Поскольку большинство серверов Интернет работают под управлением операционной системы UNIX (или ей подобной), то для разделения отдельных папок в «пути» используется прямой слеш (/), а не обратный (\), как принято в Microsoft Windows или MS DOS.*

имя файла – имя файла, в котором находится данный ресурс.

дополнительная информация – в качестве дополнительной информации могут выступать, например:

а) идентификатор фрагмента файла – для непосредственного доступа к конкретному фрагменту HTML-документа, обозначенного меткой («якорем»). В этом случае метке должен предшествовать символ «решетка» (#).

б) параметр – для передачи дополнительной информации в программу, которая вызывается в качестве ресурса на сервере в соответствии с общим интерфейсом шлюзов CGI. В этом случае параметру должен предшествовать символ «вопросительный знак» (?).

При записи URL необходимо учитывать, следующее:

- могут использоваться только буквы латинского алфавита и некоторые символы, такие, например, как «тире» (-), «подчеркивание» (_), «тильда» (~) и т.д.;
- если же необходимо использовать символы кириллицы, то каждый из них необходимо представить в Юникоде (UTF-8) в виде последовательности из двух байтов в шестнадцатеричном представлении, перед каждым из которых ставится знак «процента» (%); например, большая русская буква «М» будет записана как «%D0%9C»;

- нельзя использовать символ «пробел»;
- прописные и строчные символы, входящие в его состав, различаются (за исключением компонента **хост**, если он задан в виде доменного имени).

Ниже приведены наиболее типичные примеры записи URL реальных Web-страниц:

1. http://www.ksame.kharkov.ua/index.php – наиболее распространенный способ записи URL, в котором указаны протокол (http), доменное имя (www.ksame.kharkov.ua) и имя файла (index.php) – начальная страница сайта Харьковской национальной академии городского хозяйства.
2. www.ksame.kharkov.ua – аналог предыдущего URL, в котором опущены протокол и имя файла. Поскольку, имя компьютера (www) в доменном имени явно указывает на его принадлежность ко Всемирной Паутине, то протокол http будет подставлен автоматически. Если же не указано имя файла, то по умолчанию серверы WWW настроены на выдачу страниц типа *index.html*, *index.htm* или *index.php* – как в данном примере.
3. http://www.ksame.kharkov.ua/portal/index.php?mnu=33 – URL с указанием дополнительной информации (?mnu=33) – страница с расписанием занятий студентов на Web-портале Харьковской национальной академии городского хозяйства.
4. http://ru.wikipedia.org:80/wiki/URL – URL с явным указанием номера порта (80) – описание Единообразного локатора ресурса на сайте свободной энциклопедии Википедия.
5. ftp://ftp.microsoft.com/ – URL с ftp протоколом – «подвал» корпорации Microsoft – обширное хранилище полезных программ, предназначенных для скачивания.
6. mailto://MyMail@ksame.kharkov.ua – URL адреса (несуществующего) электронной почты – протокол mailto.
7. telnet://www.ksame.kharkov.ua – подключение к Web-сайту Харьковской национальной академии городского хозяйства по протоколу telnet (необходимы так же логин и пароль).

8. file://D:\course1\БУА-1\Иванов\HTML\index.html – URL для просмотра файла index.html, расположенного на локальном компьютере. ⚠ *Использование символов кириллицы (русских) и обратного слеша (\) вместо прямого (/) в данном примере возможно, поскольку компьютер работает под управлением локализованной версии Microsoft Windows.*

В последнее время URL позиционируется как часть более общей системы идентификации ресурсов – *URI – Uniform Resource Identifier – универсальный индикатор ресурса*. И сам термин URL постепенно уступает место более широкому термину URI.

Полное описание стандарта URL, одобренного комитетом W3C, содержится в RFC 1738.

ЛЕКЦИЯ № 6

ОСНОВЫ ВЕБ-ДИЗАЙНА

План

1. Назначение языка HTML
2. Понятие Web-страницы и Web-сайта
3. История создания языка HTML
4. Структура HTML-документа
5. Инструментарий для создания HTML-документов

1. Назначение языка HTML

Наиболее востребованным на сегодняшний день сервисом Интерната является всемирная паутина – World Wide Web (WWW). Она «соткана» из множества отдельных Web-страниц, каждая из которых состоит из:

1. *содержания*,
2. *представления* и
3. *поведения*.

Содержание Web-страницы – это текст, графика, видео и аудио – одним словом, это информация, которая помещается на страницу. Для описания структуры этой информации служит язык *HTML* (*Hyper Text Markup Language* – язык гипертекстовой разметки). Представление описывает формат вывода этой информации. Именно представление делает Web-страницы такими привлекательными. Представление Web-страниц реализуется при помощи каскадных таблиц стилей (*Cascading Style Sheets – CSS*). Поведение – это динамическое изменение Web-страницы в зависимости от действий пользователя или каких-либо иных событий. Оно задается с помощью так называемых *сценариев* (*скриптов*) – программ, которые могут быть записаны прямо в коде Web-страниц. Такие программы создаются на одном из скриптовых языков. Чаще всего – на *JavaScript*.

Основой любой Web-страницы является ее содержание – без содержания страница существовать не может. Оно описывается (размечается) при помощи языка HTML. То есть, с помощью языка HTML описываются различные фрагменты документа (текст, рисунки, гиперссылки и т. д.), а так же их взаимное расположение.

Для просмотра документов, размеченных по правилам языка HTML, предназначены специальные программы – *браузеры* (англ. *browsers*). Они читают файл HTML-документа, форматируют его в виде Web-страницы и отображают на мониторе пользователя. Существует большое разнообразие программ-браузеров от различных производителей. Однако, на сегодняшний день, наиболее распространенными являются Internet Explorer корпорации Microsoft, а также Google Chrome разработанный в корпорации Google Inc, Mozilla Firefox от Mozilla Corporation и Opera от Opera Software.

2. Понятие Web-страницы и Web-сайта

Web-страницей обычно называют отдельный HTML-документ, размеченный при помощи языка HTML и предназначенный для публикации в сети Интернет. Набор таких Web-страниц, связанных общей тематикой, называют Web-сайтом.

Одна из этих страниц является основной (головной, стартовой, индексной и т. д.) и автоматически выдается на просмотр пользователю, указавшему в браузере только имя сайта. Все остальные страницы сайта, как правило, выдаются пользователю из основной при помощи гиперссылок.

Каждый отдельно взятый Web-сайт, как правило, кроме общей тематики, отличает еще и общий дизайн, т. е. общий вид и компоновка страниц. Это позволяет посетителям сайта легко на нем ориентироваться, так как страницы выполнены в едином стиле, на всех страницах одинаковая или похожая навигация, а меняется только их содержание.

3. История создания языка HTML

Начало развития языка HTML было положено выпускником Оксфордского университета, бакалавром в области физики, Тимом Бернерс-Ли (Tim Berners Lee). В 1989 году он выдвинул предложение о *Системе гипертекстовых документов* для создания внутренней информационной службы Европейского центра ядерных исследований (CERN). В 1990 году он назвал ее World Wide Web (на русский язык это можно перевести как Всемирная Паутина). Одной из составляющих этой системы был язык гипертекстовой разметки. Основные принципы языка HTML были заложены Тимом Бернерс-Ли в том же 1990 году, когда он работал над программой первого текстового Web-браузера. За основу им был взят язык для логического оформления текста SGML (Standard Generalized Markup Language), стандарт которого был принят Международной организацией по стандартизации (ISO) в 1986 году.

А первым (и долгое время единственным) графическим браузером в те далекие времена была программа Mosaic, разработанная, как и сама World Wide Web, в научном учреждении – Национальном центре суперкомпьютерных приложений США (National Center for Supercomputer Applications – NCSA) группой программистов возглавляемой Марком Андрисеном в сентябре 1993 года. Быстроте, с которой новинка покорила виртуальное пространство, может позавидовать иной вирус – всего за год около двух миллионов пользователей установили Mosaic на свои компьютеры. Именно благодаря популярности браузера Mosaic язык HTML получил такое распространение.

В том же 1993 году появилась первая версия языка – HTML 1.0. Но поскольку она не была стандартизована, то разные производители браузеров стали предлагать свои варианты инструкций HTML, число которых все возрастало и возрастало. А для существования сети необходимо, чтобы авторы Web-страниц и производители браузеров пользовались одними и теми же соглашениями. Это и послужило причиной для начала совместной работы по созданию спецификации языка HTML. Такую работу взяла на себя организация, называемая World Wide Web Consortium (сокращенно W3C). В ее

задачу входило составление спецификации, отражающей текущий уровень развития возможностей языка с учетом разнообразных предложений компаний-разработчиков браузеров. Так, в ноябре 1995 года появилась спецификация HTML 2.0, призванная формализовать сложившуюся к концу 1994 года практику использования HTML. К тому времени новую версию языка HTML полностью поддерживало большинство браузеров. Последней версией языка, которая стала стандартом в 1999 году, является HTML 4.01. Следующая версия языка – HTML 5 – находится на стадии разработки. Однако большинство современных браузеров уже сейчас поддерживают его в том или ином виде.

4. Структура HTML-документа

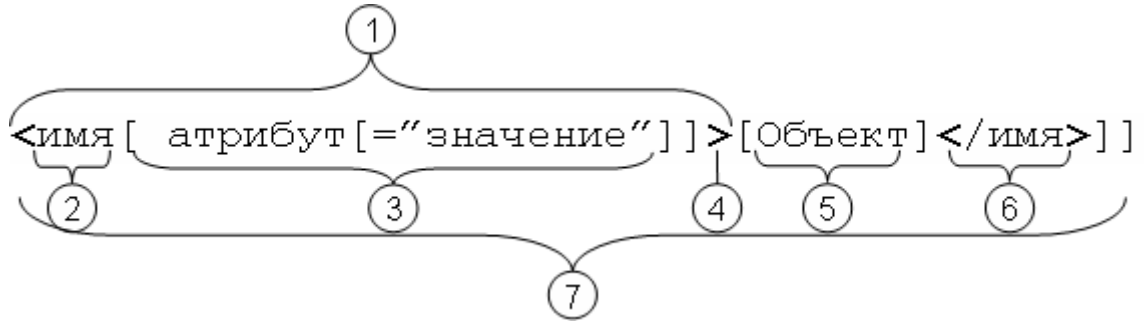
HTML-документ представляет собой обыкновенный *текстовый файл* в формате ANSI ASCII, который состоит из 2-х типов объектов:

1. объекты, которые должны отображаться на Web-странице (текст, графика, мультимедиа и т. д.) и
2. объекты, которые управляют расположением и отображением на Web-странице первых.

Объекты, которые управляют расположением и отображением на Web-страницах текста, графики и т. д., а сами на экране не отображаются, в языке HTML принято называть *тегами* (от английского слова *tag* – ярлык, признак). Теги вместе с объектами (фрагментами документа), отображением которых он управляет, называются *элементами* Web-страницы. Формальный синтаксис элемента представлен на рисунке 6.1.

Для отделения тегов от других объектов HTML-документа они заключаются в символы-ограничители «**<**» и «**>**» (поз. 4. Рис. 6.1), между которыми записываются *имя* тега (поз. 2. рисунок 6.1) и, возможно, его *атрибуты* (поз. 3. рисунок. 6.1). Имя тега указывает на характер выполняемых им действий по разметке и форматированию, а атрибуты (параметры) уточняют способ выполнения этих действий. Многие атрибуты требуют указания их значений, которые следуют за знаком равенства, стоящим после имени

атрибута. Если значение атрибута – одно слово или число, то его можно просто указать после знака равенства, не выделяя дополнительно. Все остальные значения необходимо заключать в одинарные «′» или двойные кавычки «″», особенно если они содержат несколько разделенных пробелами слов. Длина значения атрибута ограничена 1024 символами. Некоторые атрибуты значений не имеют или могут быть записаны без них. В последнем случае они принимают некоторые значения по умолчанию. Атрибуты являются не обязательными составляющими тега и если они имеются, то отделяются от имени, и друг от друга, по крайней мере, одним пробелом. Порядок следования атрибутов тега произволен.



а. Открывающий тег
б. Имя тега
в. Атрибут
г. Символ-ограничитель
д. Отображаемый объект
е. Закрывающий тег
ж. Элемент

Рисунок 6.1 – Элемент HTML-документа

Составляющие в квадратных скобках «[]» являются не обязательными.

Большинство тегов языка HTML являются контейнерами, т. е. отображаемый объект (поз. 5. рисунок 6.1) помещается между открывающим (поз. 1. рисунок 6.1) и, соответствующим ему, закрывающим (поз. 6. рисунок 6.1) тегами. Закрывающий тег имеет такие особенности:

3. имя закрывающего тега идентично имени открывающего;
4. перед именем закрывающего тега всегда ставится наклонная черта (прямой слеш) «/»;
5. закрывающий тег не имеет атрибутов (параметров).

Однако не все теги являются контейнерами – некоторые состоят только из открывающего, например, `<br>` – перевод строки. Такие элементы называются *пустыми*.

В языке HTML регистр символов в именах тегов и атрибутов не учитывается. Поэтому их можно записывать как строчными, так и прописными буквами, или их сочетанием. Например, правильно будут обработаны такие теги, как `<br>`, `<BR>` и `<Br>`. Однако, при записи значений атрибутов регистр символов следует учитывать.

Среди множества тегов языка HTML есть несколько *обязательных*, т. е. тех, которые всегда должны присутствовать в любом *действительном* HTML-документе. Их немного – всего четыре: `<html>`, `<head>`, `<title>` и `<body>` – вместе они создают структуру HTML-документа (рисунок 6.2).

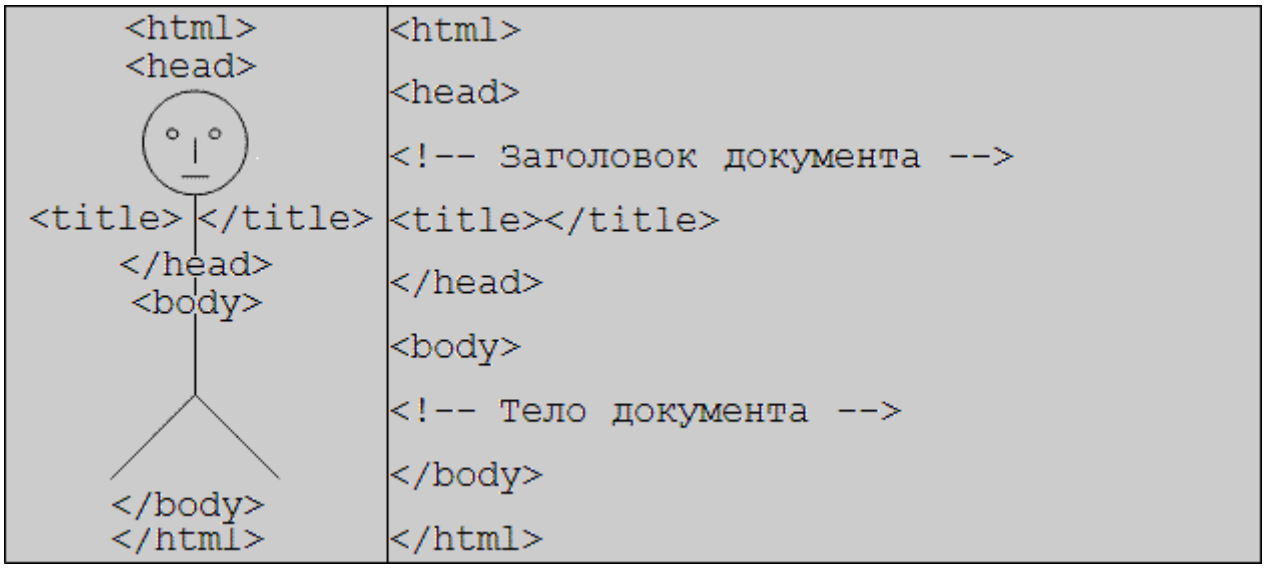


Рисунок 6.2 – Структура HTML-документа

Любой HTML-документ – это большой html элемент-контейнер. Его первым, открывающий, тегом должен быть `<html>` и, соответственно, последним, закрывающим – `</html>`. Этот контейнер, в свою очередь, содержит два элемента:

1. *h e a d* (заголовок), который оформляется при помощи тегов `<head>` и `</head>`,
2. *b o d y* (тело), которое оформляется при помощи тегов `<body>` и `</body>`.

В заголовке HTML-документа содержится в основном служебная информация о нем, тогда как тело документа содержит непосредственно сам документ – текст, рисунки и прочие элементы, которые отображаются на экране. Элементы, находящиеся внутри заголовка, браузером не отображаются.

Исключение составляет элемент `title` – название документа, которое отображается в строке заголовка браузера.

Поскольку большинство тегов языка HTML являются контейнерами, то помимо отображаемых объектов они могут содержать и другие теги, т. е. теги могут быть вложенными друг в друга. При этом необходимо соблюдать определенный порядок открывающих и закрывающих тегов: *тег, который был открыт первым, закрывается последним, а последний – первым*. То есть, такая последовательность тегов:

<тэг$_1$><тэг$_2$> ... </тэг$_2$></тэг$_1$> – правильная,

а такая:

<тэг$_1$><тэг$_2$> ... </тэг$_1$></тэг$_2$> – нет.

HTML-документ помимо тегов может содержать так же *комментарии*. Они ни как не влияют на отображение документа, а служат для пояснения отдельных его фрагментов. Например, если над документом работает несколько человек, или для того, чтобы через некоторое время было легче вспомнить и понять свой собственный текст. Комментарии так же иногда используются, когда необходимо временно исключить некоторый фрагмент кода из обработки, не удаляя его совсем. Для этого необходимо заключить такой фрагмент в комментарии, после чего браузером он будет игнорироваться.

Хотя текст комментариев и не отображается браузером, однако передается вместе с документом и вполне может быть просмотрен. Так происходит потому, что большинство браузеров предоставляют возможность просмотра исходного кода документа. Поэтому не следует включать в комментарии информацию, не предназначенную для чужих глаз.

Комментарии в языке HTML имеют следующий синтаксис:

```
<!-- Текст комментария -->
```

То есть, синтаксически это пустой элемент, имя которого «`!--`», а семантически (по смыслу) он, конечно же, отличается от любого тега.

5. Инструментарий для создания HTML-документов

Для создания любого документа нужна соответствующая программа, и Web-страницы не являются исключением. Существует различные способы создания таких страниц. Простейший из них состоит в том, что некоторые программы (такие, например, как Microsoft Word, Microsoft Excel, OpenOffice.org Writer, OpenOffice.org Calc, LibreOffice Writer, LibreOffice Calc и д. р.) «умеют» сохранять свои документы в формате .html. Для этого необходимо:

1. подготовить нужный документ,
2. выполнить команду **Файл ⇨ Сохранить как** и
3. в раскрывшемся окне диалога указать, что файл необходимо сохранить в формате HTML-документа.

Основные достоинства этого метода состоят в том, что Web-страницы создаются довольно быстро и просто, а так же совершенно не требуется знание языка HTML. Но на этом достоинства этого метода и заканчиваются – начинаются его недостатками. Наиболее существенные из них состоят в том, что страницы получаются «массового производства», их размеры в десятки раз превышают аналогичные, но созданные «вручную», изменять такие страницы в дальнейшем весьма проблематично. Соответственно, этот способ используется, в основном, в экстренных случаях – т. е. когда катастрофически не хватает времени, а срочно необходимо один раз быстро создать несколько страниц, которые в дальнейшем не подлежат изменению.

Более совершенный способ создания Web-страниц состоит в использовании специальных программ – HTML-редакторов, которые бывают двух видов:

1. *визуальные* и
2. *текстовые*.

При использовании визуальных редакторов знания языка HTML является не обязательным. Они позволяют работать с Web-страницей «как она есть». То есть, пользователь редактирует и форматирует текст, вставляет рисунки,

таблицы, как в обычном текстовом редакторе, а уж сама программа генерирует соответствующий HTML-код. Именно поэтому визуальные редакторы еще называют WYSIWYG-редакторами. Аббревиатура WYSIWYG расшифровывается как What You See Is What You Get – «что видишь, то и получишь». Следует отметить, что Microsoft Word (или OpenOffice.org Writer, или LibreOffice Writer), по существу, тоже является WYSIWYG-редактором, хотя и «прячут от пользователя» свои команды разметки и форматирования внутри текстового документа.

К наиболее распространенным визуальным редакторам относятся, например, Microsoft FrontPage, Macromedia DreamWeaver, HotDog от Sausage Software и некоторые другие. Они имеют мощные средства визуального редактирования, одновременно предоставляя доступ к созданному HTML-коду. Таким образом, работа в этих редакторах может состоять из 2-х шагов:

1. с начала визуальном режиме создается нужный HTML-документ,
2. а затем, в случае необходимости, можно произвести «более тонкую» его настройку, редактируя полученный на первом шаге код.

Использование визуальных HTML-редакторов существенно экономит время и силы разработчиков Web-страниц.

Однако следует заметить, что ни один визуальный редактор не совершенен, и все они, так или иначе, ограничены в своих возможностях. Используя подобный редактор очень трудно создать что-либо оригинальное и эффектно смотрящееся, поскольку фантазия автора всегда будет ограничена рамками конкретной программы. А любому человеку всегда будет приятнее получить вещь, созданную мастером специально для него, чем приобрести аналог фабричного производства, который имеется дома у каждого второго. Поэтому для создания не «штампованных», оригинальных Web-страниц их код пишется руками при помощи текстовых HTML-редакторов, которые еще называются *редакторами тегов*.

Как правило, текстовые HTML-редакторы создаются небольшими фирмами или отдельными программистами, большинство из которых

распространяется совершенно бесплатно. Среди таких редакторов есть совсем простые – например, WebCoder, созданный Дмитрием Захаровым. Редакторы тегов такого уровня, как правило, имеют различные функции, облегчающие написание кода. Так, например, при создании нового документа они позволяют быстро создавать заготовку будущей Web-страницы; по нажатию различных «горячих клавиш» и кнопок вставлять уже готовые конструкции (куски кода, спецсимволы и т. д.); автоматически вызывать браузер для просмотра редактируемого документа и т. п. Большинство из них так же поддерживает подсветку синтаксиса, т. е. HTML-код отображается различными цветами. Например, теги выделяются одним цветом, атрибуты – другим, их значения – третьим, а непосредственно текст страницы – четвертым. Это значительно упрощает чтение и редактирование кода – открыв документ, легко можно найти по цвету нужный элемент и отредактировать его.

Есть и более совершенные текстовые HTML-редакторы, такие, например, как Macromedia HomeSite, HTMLPad FisherMan, HTMLPad 2007, QuickHTML, NotePad++ (только для Windows), Quanta Plus (для UNIX-подобных операционных систем), Bluefish, Geany (есть версии для UNIX-подобных операционных систем, и для Windows) и многие, многие другие. Они позволяют не только создавать довольно сложные элементы, такие, например, как таблицы, но и работать над проектами – наборами взаимосвязанных файлов, образующих сайт.

Между тем, поскольку HTML-документ является обычным текстовым файлом, то для его создания подойдет любой текстовый редактор. Простейший вариант – программа Блокнот, которая имеется у каждого пользователя, поскольку входит в стандартную поставку операционной системы Microsoft Windows. Она является наилучшим вариантом на начальном этапе освоения языка HTML – *с программы Блокнот когда-то начинали все создатели Web-сайтов*. Программа Блокнот всегда запустится по команде основного меню Windows **Пуск ⇨ Все программы ⇨ Стандартные ⇨ Блокнот**. Если при

этом в нем набрать заготовку Web-страницы, представленную на рисунок 6.2, то она будет выглядеть следующим образом (рисунок 6.3):

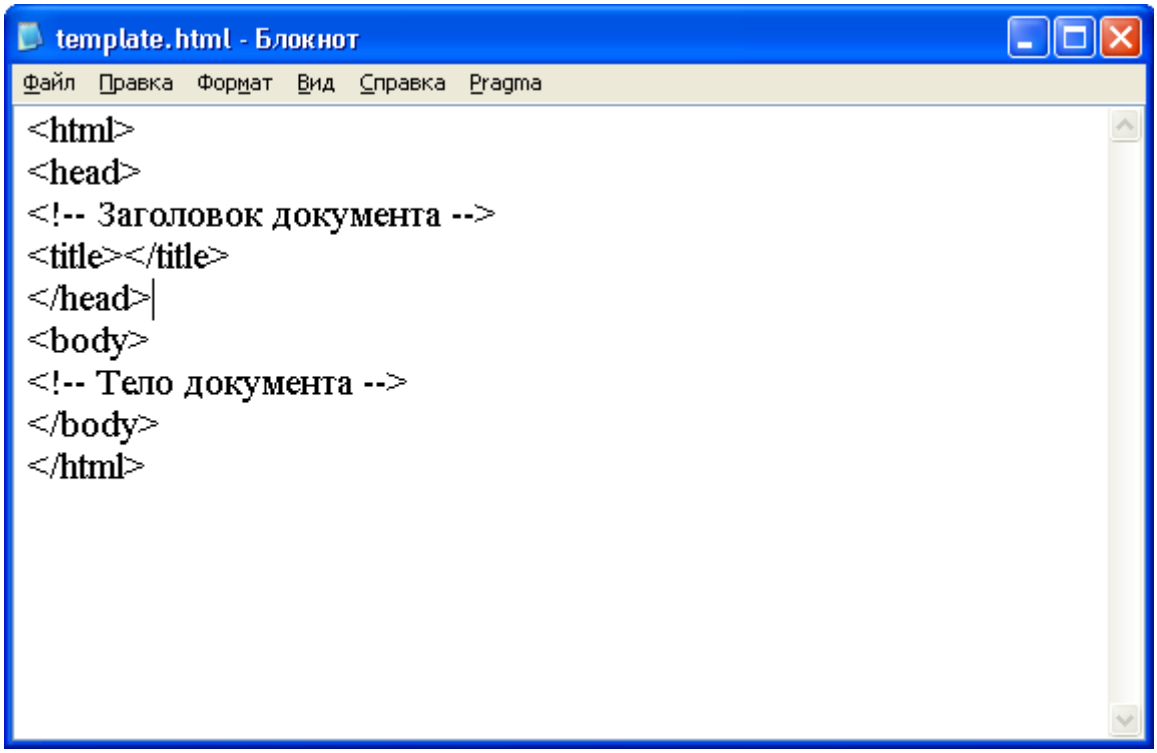


Рисунок 6.3 – Заготовка Web-страницы в программе Блокнот

⚠ *При первом сохраненииHTML- документа в окне диалога* ***Сохранить*** *необходимо явно указать:*

1. *тип файла –* ***Текстовый документ,***
2. *имя файла – латинскими буквами и*
3. *расширение имени файла – .html или .htm.*

В дальнейшем необходимо привыкать к латинским буквам в именах файлов, поскольку не все компоненты Интернет адекватно работают с русскими символами.

Если теперь двойным щелчком открыть созданный HTML-документ в браузере, например, Microsoft Internet Explorer, то будет выведена пустая страница (Рисунок 6.4):

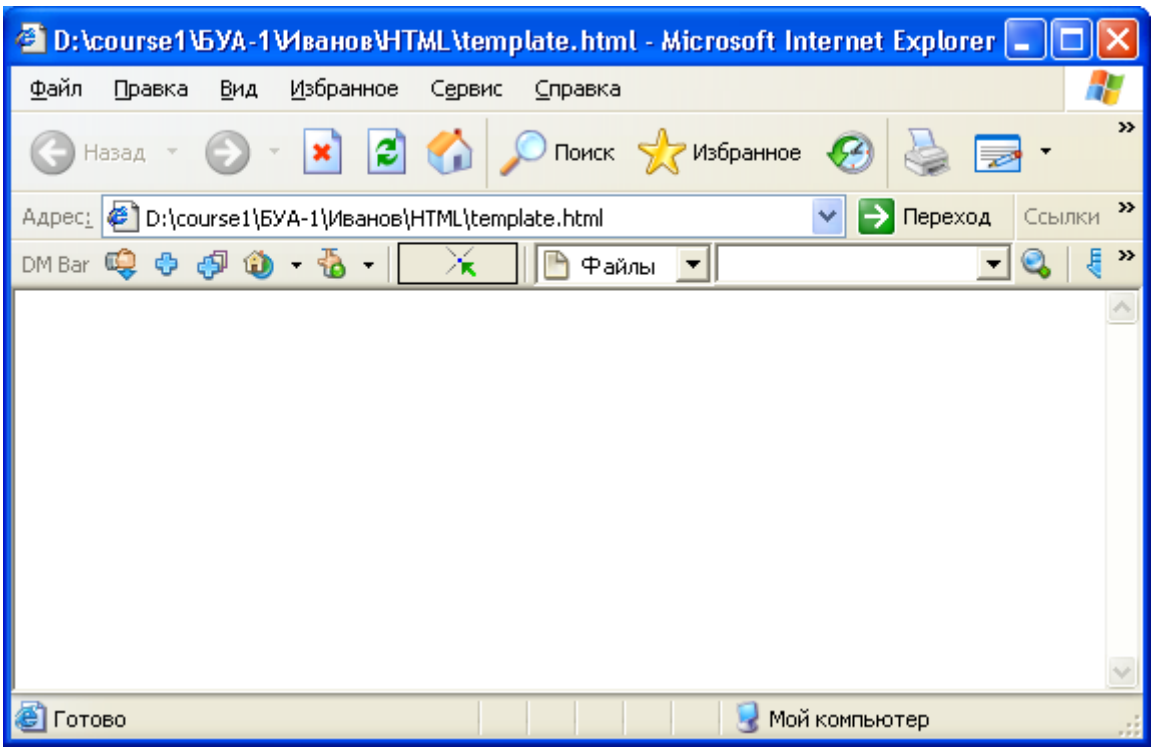


Рисунок 6.4 – Заготовка Web-страницы в браузере Microsoft Internet Explorer

Для того, чтобы Web-страница приняла более информативный вид, необходимо в заготовку HTML-документа внести некоторые изменения. Например, между тегами `<title >` и `</title>` поместить текст «Мой первый документ HTML», а между `<body>` и `</body>` – «Всем огромный привет!». Тогда откорректированный текст HTML-документа должен выглядеть следующим образом:

```
<html>
<head>
<!-- Заголовок документа -->
<title>Мой первый документ HTML</title>
</head>
<body>
<!-- Тело документа -->
Всем огромный привет!
</body>
</html>
```

⚠*Для того, чтобы увидеть внесенные в HTML-документ изменения необходимо:*

1. *сохранить его в программе Блокнот (выполнив, например, команду **Файл ⇨ Сохранить**) и*
2. *обновить содержание окна браузера (щелкнув по кнопку **Обновить**).*

В результате откорректированный HTML-документ должен выглядеть как на рисунке 6.5.

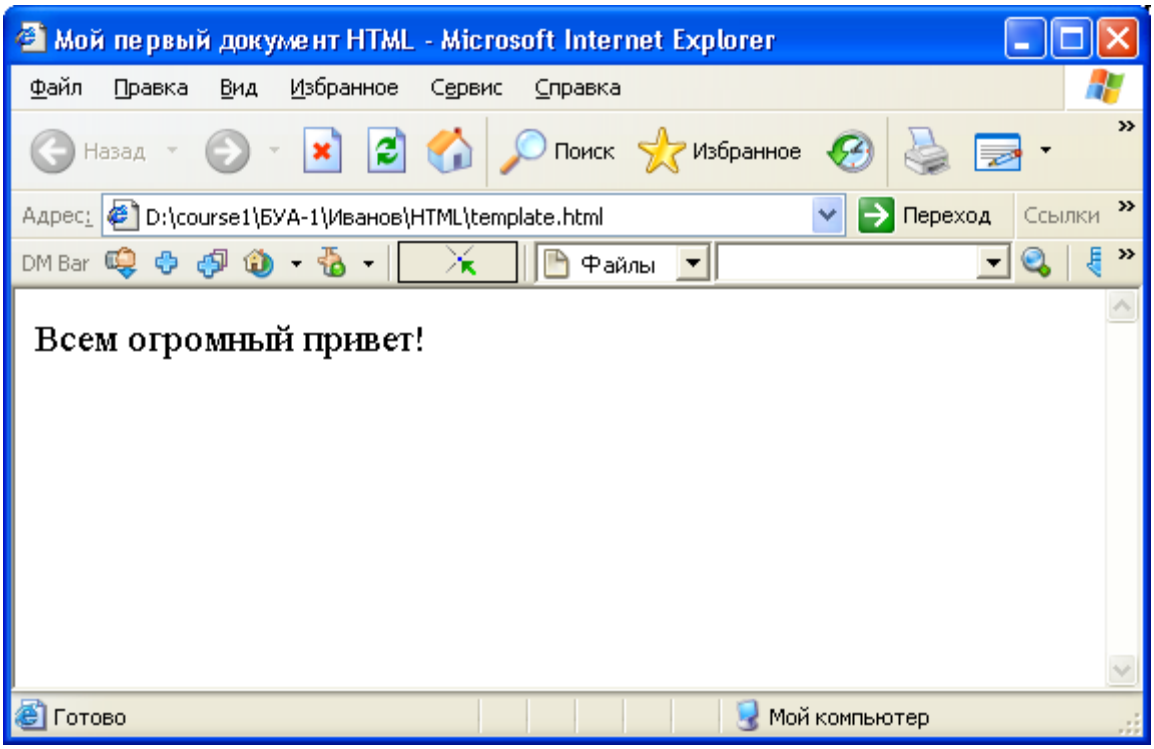


Рисунок 6.5 – Первый HTML-документ

Приведенные выше текст Web-страницы был разбит на строки только лишь с целью лучшей читабельности и восприятия человеком. Большинство современных браузеров воспринимают код HTML-файлов как одну длинную строку, пропуская при этом все лишние пробелы, символы конца строк и табуляции. То есть, «с точки зрения браузера» этот текст выглядит следующим образом:

```
<html><head><!-- Заголовок документа --><title>Мой первый документ HTML</title></head><body><!-- Тело документа -->Всем огромный привет!</body></html>
```

Если этот текст так же (без разбивки на строчки) набрать в Блокноте, то внешний вид Web-страницы в окне браузера при этом не изменится.

ЛЕКЦИЯ № 7

ПРОГРАММНЫЕ СРЕДСТВА РАБОТЫ СО СТРУКТУРИРОВАННЫМИ ДОКУМЕНТАМИ

План

1. Основная идея электронных таблиц
2. История создания электронных таблиц
3. Организация данных в электронных таблицах
4. Адресация ячеек
5. Относительная и абсолютная адресация
6. Особенности интерфейса Microsoft Excel
7. Режимы работы
8. Завершение работы

1. Основная идея электронных таблиц

Электронные таблицы (табличные процессоры) – это программы, которые предназначены для обработки информации, представленной в табличном виде, т. е. в виде строчек и столбцов, на пересечении которых находятся ячейки.

Основная идея электронных таблиц состоит в том, что одна часть ее ячеек содержит исходные данные, а другая – формулы, по которым эти данные пересчитываются. В результате этого всякое изменение содержимого какой-либо ячейки исходных данных приводит к немедленному изменению содержимого ячеек с формулами – т.е. содержимое электронной таблицы обновляется (пересчитывается) автоматически, и всегда остается актуальным.

Столь простая, и в тоже время, абсолютно гениальная идея послужила толчком к созданию огромного количества программ этого класса – в мире, наверное, нет ни одного компьютера, на котором бы не использовалась та, или иная, электронная таблица. Наиболее широкое распространение они получили в области выполнения различного рода экономических расчетов, поскольку

являются естественным продолжением идеи хорошо известных всем экономистам «пустографок». При этом переход от традиционной, «бумажной пустографки», к ее электронному эквиваленту происходит совершенно легко и естественно. А однажды заполнив такую «электронную пустографку» исходными данными и формулами в дальнейшем ее можно только «эксплуатировать» – заносить исходные данные, а результаты будут получаться автоматически, и притом без всяких ошибок усталости и невнимательности, свойственных ручному счету. Однако область применения электронных таблиц вовсе не ограничивается только экономическими расчетами. Благодаря наличию в них большого количества самых разнообразных функций, а также средств наглядного представления информации в виде различного рода графиков и диаграмм, они нашли самое широкое применение и в других областях выполнения расчетов, таких, например, как инженерные, научно-технические и т. д. Если сравнить электронную таблицу с таким широко распространенным инструментом для выполнения вычислений как калькулятор, то можно смело сказать, что последний выполняет только *арифметические* вычисления, в то время как электронная таблица – также и *алгебраические*, в том смысле что она может оперировать не только *константами*, как калькулятор, но и *переменными* – содержанием ячеек.

2. История создания электронных таблиц

Самая первая электронная таблица – *VisiCalc* – была создана в 1979 году Дэниелом Бриклином и Робертом Фрэнкстоном и принесла своим создателям только в течение первого года продаж около двух миллионов долларов чистой прибыли. А началось все с того, что опыт работы в компьютерной индустрии выпускника Массачусетского технологического института (МТИ) Дэниела Бриклина навел его на мысль, что, не плохо разбираясь в компьютерах, он при этом совершенно ни чего не понимает в бизнесе. Это и побудило его пойти на курсы в Гарвардскую школу бизнеса. Во время занятий на этих курсах ему приходилось выполнять много однообразных вычислений, основная специфика

которых заключалась в том, что при изменении какой-либо одной цифры необходимо было пересчитывать все, зависящие от нее, величины. Малейшая ошибка при этом могла испортить всю работу, которая записывалась на большие листы тщательно разлинованной бумаги – *Spreadsheets* – развернутые листы, или по-русски – «простыни» «пустографки». При этом программистский опыт Бриклина натолкнул его на мысль, что все эти однообразные и нудные жонглирования цифрами неплохо было бы «поручить» компьютеру. Своими соображениями он поделился с Робертом Фрэнкстоном, которого эта идея тоже вдохновила, и они вместе приступили к ее реализации. В результате к весне 1979 года идея была воплощена в законченную программу, которую ее создатели назвали VisiCalc (Visible Calculator – визуальный калькулятор). Изначально она была написана для компьютера Apple-2 и в течение первого года после начала продаж (с августа 1979 года) разошлась тиражом около 100 тыс. экземпляров по цене 200 долл. за штуку.

Первая же электронная таблица для компьютеров типа IBM PC была создана Митчелом Кэпором спустя три года – в конце 1982 году – и называлась *Lotus 1-2-3*. При этом на ее рекламу было истрачено 1 млн. долл. Но эти расходы окупились очень скоро – уже через полгода было продано 60 тыс. копий по цене 495 долл. за штуку. Одним из наиболее распространенных табличных процессоров в предшествующие десятилетия на территории бывшего Советского Союза был SuperCalc фирмы Computer Associates – «облегченный», по сравнению с Lotus 1-2-3, вариант электронных таблиц для операционной системы MS-DOS.

В настоящее время, ведущее положение на мировом рынке программных продуктов этого класса занимает электронная таблица *Excel* корпорации Microsoft, первая версия которой была выпущена в 1985 году. В 1993 году появилась 5-я версия Excel, ставшая первым приложением знаменитого пакета Microsoft Office. В дополнение к традиционным табличным вычислениям, в основном, в области экономики и финансов, Microsoft Excel позволяет строить различные диаграммы, графики и рисунки, проводить сложную статистическую

и математическую обработку данных, обмениваться информацией с наиболее распространенными базами данных, создавать Web-страницы и т.д. Средствами табличного процессора Microsoft Excel можно производить полный цикл обработки информации – от сбора, регистрации и, собственно, обработки до отображения ее в наиболее презентабельном виде, достигая при этом поистине ошеломляющих результатов.

3. Организация данных в электронных таблицах

Любая программа, и электронные таблицы в том числе, может обрабатывать данные только вполне определенного вида (формата, структуры и т.д.). Основными объектами данных, с которыми может работать электронная таблица, являются следующие:

1. *рабочая книга*,
2. *лист*,
3. *строка*,
4. *столбец*,
5. *ячейка*,
6. *диапазон* и
7. *список*.

Документ (т. е. объект обработки) приложения Microsoft Excel называется *рабочая книга*. Каждая рабочая книга имеет собственное имя, построенное в соответствии со стандартами операционной системы Windows, и хранится в отдельном дисковом файле. По умолчанию, например, новым рабочим книгам Microsoft Excel присваивает имена **Книга1**, **Книга2** и т. д. При их записи на диск к этому имени по умолчанию добавляется расширение **.XLS**, так что на диске будут записаны, соответственно, файлы **Книга1.XLS**, **Книга2.XLS** и т. д. В электронных таблицах допускается одновременная работа с несколькими рабочими книгами, при чем каждая из них открывается в отдельном окне документа. Количество одновременно открытых рабочих книг

ограничивается только наличием свободных ресурсов компьютера. При этом *активной*, или *текущей*, может быть только одна из них.

Каждая рабочая книга состоит из *листов* двух типов:

1. *Рабочий лист* (или просто лист) образует основное рабочее пространство пользователя, которое может состоять из таких компонентов:
   а) *ячеек*, организованных в виде *строк* и *столбцов*,
   б) одной или нескольких диаграмм,
   в) стандартных элементов управления, таких как кнопки, флажки, переключатели и т. д. и
   г) прочих связанных и внедренных объектов из других приложений.

   Все компоненты, кроме а), необязательны и могут отсутствовать на конкретном рабочем листе – они туда помещаются явным образом. По умолчанию рабочим листам присваиваются имена **Лист1**, **Лист2** и т. д.
2. *Лист диаграмм* предназначен для размещения на нем диаграмм. На одном листе диаграмм может размещаться одна, или несколько, диаграмм. Диаграммы можно свободно перемещать между листами диаграмм и рабочими листами в любом направлении. Стандартные имена листов диаграмм – **Диаграмма1**, **Диаграмма2** и т. д.

Количество листов в рабочей книге может быть произвольным – оно ограничено только наличием свободных ресурсов на компьютере. По умолчанию каждая, вновь создаваемая рабочая книга, содержит три рабочих листа – **Лист1**, **Лист2** и **Лист3**.

Каждый рабочий лист состоит из 65 536 = 2^{16} пронумерованных *строк*, начиная с 1, и 256 = 2^8 *столбцов*, проиндексированных буквами *латинского алфавита*, начиная с A, и заканчивая IV. На пересечении строк и столбцов находятся *ячейки*, общее количество которых – 16 777 216 = 65 536 × 256.

 *Ячейка – это элементарная (неделимая) и однозначно адресуемая единица информации, обрабатываемой с помощью электронной таблицы*.

Каждая ячейка рабочего листа может быть:

1. *пустая*, или

2. *непустая*.

Непустая ячейка, в свою очередь, может содержать:

а) *константу*, или

б) *формулу*.

Константы бывают:

1. *Числовые*, которые могут состоять только из цифр и, возможно, некоторых специальных знаков, таких как + - ( ) , / $ % . E e.
2. *Текстовые* – это константы, которые Excel не смог распознать как константы другого типа. Они могут содержать любые символы из допустимого набора, длиной не более 32 767 знаков. Текстовыми константами могут быть и числа, если при вводе им предшествует апостроф, например, '1234. Это могут быть, например, табельные номера сотрудников. Естественно, что такие константы не могут участвовать в арифметических вычислениях.
3. *Логические* – могут принимать только одно из двух взаимоисключающих значений: **ИСТИНА** или **ЛОЖЬ**. Они используются как индикатор наличия или отсутствия какого-либо признака или события, а также могут являться параметрами или возвращаемыми значениями некоторых функций. Во многих случаях вместо этих значений используются цифры 1 и 0, соответственно.
4. *Ошибки* – служат для индикации некоторых нештатных ситуации, возникших во время работы Excel, например деление на нуль. Значения этого типа констант начинаются с символа «#». Более детальную информацию о причине возникновения конкретной ошибки можно получить в справочной системе Microsoft Excel, осуществив поиск по типу отображаемой ошибки.

Кроме перечисленных выше основных типов данных Microsoft Excel поддерживает еще два производных:

5. *Дата и время суток* (или *Календарный*) – это тот же числовой тип данных, в котором целая часть используется для хранения количества дней прошедших с 1 января 1900 года, а дробная – это часть суток, прошедшая с полуночи.
6. *Массив*, собственно, не являются конкретным типом данных, а только образует организованное множество ячеек или констант любого типа. В Microsoft Excel массив рассматривается как единый элемент, к которому в целом могут быть применены математические, логические и другие типы операций.

Формулы в электронных таблицах служат для получения новых значений, т. е. являются основным инструментом преобразования информации. Всякая формула состоит из 2-х частей:

1. *знака равно* (=),
2. за которым следует *выражение*.

То есть, синтаксис формулы можно представить следующим образом:

```
=<выражение>
```

и означает, что результат вычисления выражения будет присвоен в качестве *значения* ячейке, в которую эта формула записана. Выражение, в свою очередь, состоит из:

1. *констант*,
2. *адресов (или ссылок на) ячейки*,
3. *функций*,
4. соединенных *знаками операций*.

При этом *круглые скобки* в выражении позволяют изменить стандартный (естественный) порядок выполнения операций. Например, формула `= 1` присваивает в качестве значения ячейки, в которой она записана, константу числового типа 1, а `= A1 + 1` – записывает в текущую ячейку значение ячейки `A1`, увеличенное на `1`.

Кроме констант или формул с любой ячейкой может быть также связано *примечание* – дополнительный текст поясняющий, например, ее содержание. Индикатор наличия примечания – маленький красный треугольник в верхнем правом углу ячейки. При перемещении указателя мыши в пределы ячейки, содержимое примечания отображается в отдельном окне примечания.

4. Адресация ячеек

Для использования значений, над которыми требуется выполнить некоторые вычисления, необходимо указать расположение ячеек, которые содержат эти значения. Расположение ячеек в электронных таблицах однозначно определяется их *адресами* (*ссылками*, *координатами* и т. д.). Для адресации ячеек в пределах одного рабочего листа используется два *стиля адресации*, которые, соответственно, называются:

1. **A1** – указывается имя столбца и номер строки,
2. **R1C1** – указывается символ R (от английского Row – строка), номер строки, символ C (от английского Column – столбец) и номер столбца.

Например, ячейка на пересечении 3-й строки и 4-го столбца в стиле **A1** имеет адрес `D3`, а в стиле **R1C1** – `R3C4`. *По умолчанию в Microsoft Excel используется стиль адресации ячеек* ***A1***. Стиль же ссылок **R1C1** довольно часто используется в *макросах*. Для того чтобы включить или выключить стиль ссылок **R1C1**, необходимо установить или сбросить, соответственно, флажок **Стиль ссылок R1C1** на вкладке **Общие** диалогового окна **Параметры**, которое раскрывается по одноименной команде из меню **Сервис**.


При изменении стиля ссылок Microsoft Excel автоматически производит соответствующие изменения в ссылках на ячейки во всех формулах рабочей книги. Визуальным признаком того, что Excel использует стиль адресации ячеек ***R1C1*** *является то, что в качестве заголовков столбцов используются цифры, а не буквы латинского алфавита, как в стиле* ***A1***.

Кроме этого, в Excel можно задавать ссылки на ячейки других листов той же книги, другой книги, а также на данные из других приложений.

*Ссылки на ячейки других рабочих книг называются **внешними**, ссылки на данные из других приложений – **удаленными**.*

Ссылки на ячейки в пределах рабочей книги включают имя листа и адрес ячейки на листе. Имя рабочего листа отделяется от адреса ячейки символом восклицательного знака (!).

Если имя листа содержит пробелы или начинается с цифры, оно заключается в одиночные апострофы, в противном случае апострофы не используются.

Например:

Накладная!D3 – адрес ячейки D3 на рабочем листе Накладная.

При необходимости анализа данных из одних и тех же ячеек на нескольких листах одной и той же рабочей книги в Microsoft Excel используются так называемые *трехмерные ссылки*. Трехмерная ссылка включает в себя ссылку на ячейку, или диапазон ячеек, перед которой указывается диапазон имена листов.

В трехмерной ссылке Excel использует все рабочие листы, указанные между первым и последним.

Например, формула

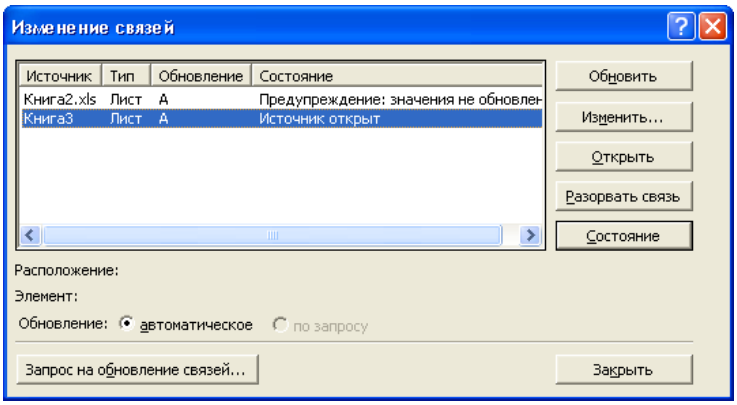
=СУММ(Лист2:Лист5!A1)

суммирует все значения, содержащиеся в ячейке `A1` на всех рабочих листах от `Лист2` до `Лист5` включительно.

В Microsoft Excel можно создавать ссылки не только между различными листами одной и той же книги, но и организовать иерархию связанных книг.

Если ячейки, на которые имеются ссылки, изменяются, Excel автоматически обновляет ссылки только для открытых книг, содержащих связи. Если же зависимая книга закрыта, связи можно обновить вручную, выполнив следующие действия:

1. Выполнить команду **Правка ⇨ Связи**. (*Недоступность пункта **Связи** означает, что данная рабочая книга не содержит связанных данных.*)
2. В раскрывшемся при этом окне диалога **Изменение связей**:



   а) В списке выбрать источник для связанного объекта. Для выделения нескольких связей необходимо выбрать их при нажатой клавише **CTRL**.

   б) Щелкнуть по кнопке **Обновить**.

Microsoft Excel отображает ссылками в формулах на другие книги двумя способами, в зависимости от состояния исходной книги (той, что предоставляет данные для формулы):

а) *открыта* она или

б) *закрыта*.

Внешний адрес ячейки в *открытой* рабочей книге содержит:

1. имя рабочей книги,
2. имя рабочего листа и
3. адрес ячейки.

Имя рабочей книги берется в квадратные скобки, имя листа следует за именем книги без разделителей, которое, в свою очередь, отделяется от адреса ячейки восклицательным знаком, например:

[Товары.XLS]Накладная!D3 – адрес ячейки D3 на рабочем листе Накладная открытой рабочей книги Товары.XLS.

Когда книга *закрыта*, ссылка должна включать дополнительно полный путь. Например:

’C:\Мои документы\[Товары.XLS]Накладная’!D3 – адрес ячейки D3 на рабочем листе Накладная рабочей книги Товары.XLS, которая находится в папке Мои Документы на диске C.


Если имя рабочего листа, книги или путь содержит символы, не являющиеся буквами, необходимо заключить их в одиночные кавычки.

5. Относительная и абсолютная адресация

Оба способа адресации указывают на одни и те же ячейки. Их различие проявляется только при копировании ячеек, содержащих формулы.

Относительная адресация основана на том, что ссылки на ячейки с исходными данными создаются относительно ячейки, содержащей формулу. Это означает, что при копировании формулы в другие ячейки ссылки в каждой копии изменяются таким образом, чтобы сохранились те же соотношения адресов, что и в исходной формуле. То есть, в формуле учитывается «расстояние» между ячейкой, содержащей формулу, и ячейкой, на которую в этой формуле есть ссылка.

При *абсолютной* адресации ссылка на ячейку содержит букву столбца и номер строки, перед которыми стоит знак доллара ($). При этом предполагается, что в результате копирования или изменении структуры рабочего листа ссылка не изменяется, и будет указывать на ту же, что и ранее, ячейку.

Смешанная адресация, т.е. комбинация относительной и абсолютной предполагает «фиксацию» только одной из двух «координат» ячейки. Циклическое изменение способа адресации во время ввода или редактирования

с относительного на абсолютный и, далее, на смешанный выполняется путем последовательных нажатий на клавишу **F4**.

Результаты выполнения копирования ячейки, в которую записана простейшая формула (например, =E7), при различных способах адресации представлены на следующих рисунках.

=C5	=D5	=E5	=F5	=G5
=C6				=G6
=C7		=E7		=G7
=C8				=G8
=C9	=D9	=E9	=F9	=G9

=$E$7	=$E$7	=$E$7	=$E$7	=$E$7
=$E$7				=$E$7
=$E$7		=$E$7		=$E$7
=$E$7				=$E$7
=$E$7	=$E$7	=$E$7	=$E$7	=$E$7

=C$7	=D$7	=E$7	=F$7	=G$7
=C$7				=G$7
=C$7		=E$7		=G$7
=C$7				=G$7
=C$7	=D$7	=E$7	=F$7	=G$7

=$E5	=$E5	=$E5	=$E5	=$E5
=$E6				=$E6
=$E7		=$E7		=$E7
=$E8				=$E8
=$E9	=$E9	=$E9	=$E9	=$E9

⚠ *По умолчанию в стиле* ***A1*** *используется относительная адресация, в то время как в* ***R1C1*** *– абсолютная*.

При работе с электронными таблицами довольно часто возникает ситуация, когда с некоторой совокупностью ячеек необходимо работать как с единым целым, например при их удалении, копировании, перемещении и т.д. В этом случае применяются так называемые *диапазоны ячеек*. Для указания диапазона ячеек используется символ двоеточия (:), помещенный между ссылками на первую и последнюю ячейки диапазона, например, B2:D4. *Диапазон может включать только смежные ячейки таблицы*. Чтобы ссылка указывала на целые столбцы или целые строки, следует использовать диапазон, содержащий в обеих частях имена столбцов или номера строк, например, B:C, 3:5. Такое обозначение диапазона позволяет автоматически включать в обработку все ячейки данных строк или столбцов, даже если впоследствии возникнет необходимость в добавлении или удалении последних. *Вместо символа двоеточия при вводе можно использовать и символ точки (.)*.

Для разделения элементов *списка*, состоящего из отдельных ячеек, и (или) диапазонов ячеек, используется символ «Разделитель элементов списка», например, 2:3,B5.

⚠ *В разных версиях и вариантах локализации операционной системы Microsoft Windows символ «Разделитель элементов списка» может иметь различное значение*.

Так, например, в русифицированной версии операционной системе Windows XP по умолчанию – это символ точки с запятой (;), а в панъевропейской – запятая (,). Его установка осуществляется по такому алгоритму:

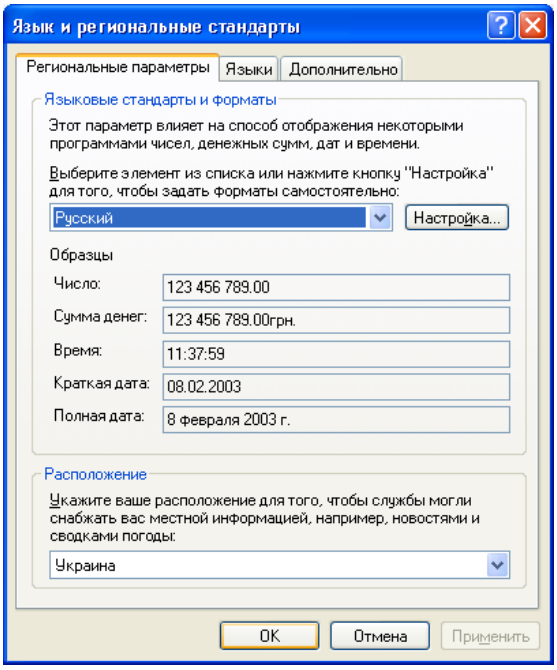
1. Выполнить команду **Пуск ⇨ Панель управления**.
2. В раскрывшемся в результате окне **Панель управления**, при отображении его:
   а) в классическом виде выбрать значение **Языки и региональные стандарты**, а

б) с разбивкой по категориям возможностей – **Дата, время языки и региональные стандарты**.

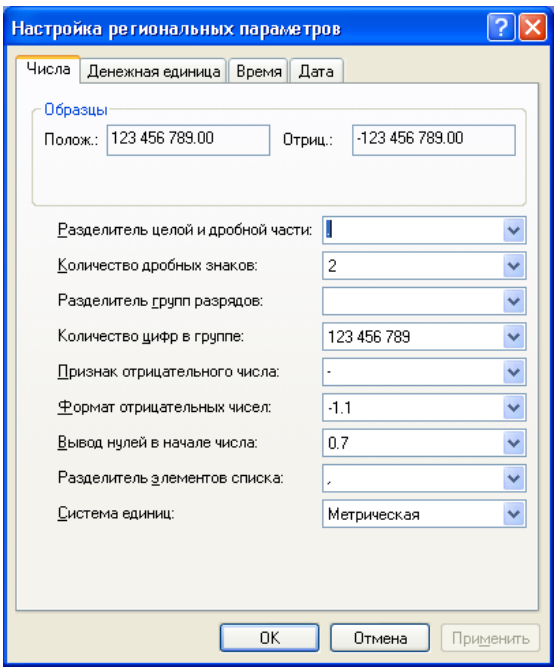
б.1) В раскрывшемся окне **Дата, время языки и региональные стандарты** выбрать:

- ❒ **Изменение формата отображения чисел, даты и времени**, или
- ❒ **Языки и региональные стандарты**.

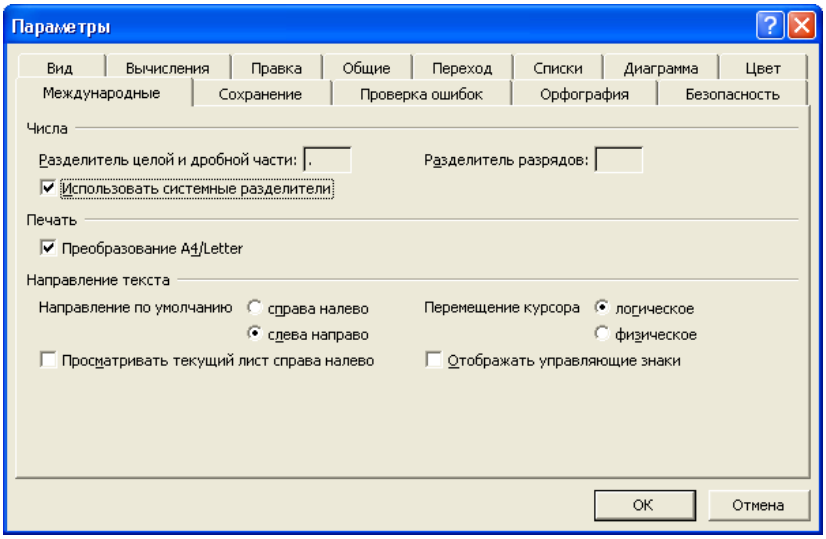
3. В окне диалога **Языки и региональные стандарты** нажать кнопку **Настройка**.



4. В поле со списком **Разделитель элементов списка** окна диалога **Настройка региональных параметров** указать необходимый символ-разделитель элементов списка.

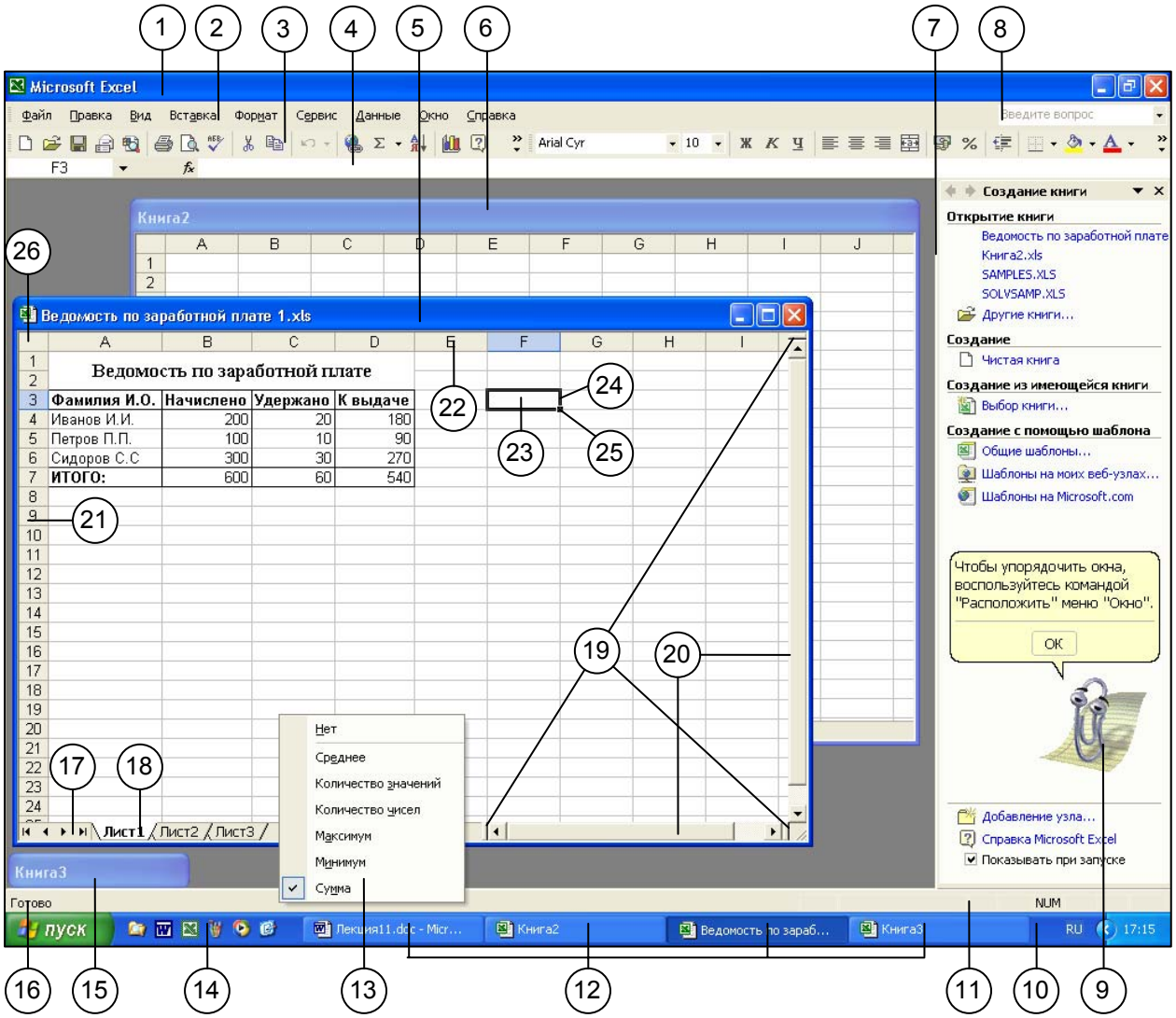


В этом окне диалоговом можно установить так же символ-разделитель целой и дробной части числа, количество знаков в дробной части числа, количество цифр в группе, и другие параметры отображения числовых величин, даты, времени, а так же денежных единиц для операционной системы Windows в целом. Отказаться от использования системных (глобальных) разделителей, и определить свои, можно на вкладке **Международные** окна диалога **Параметры**, которое открывается по команде **Сервис ⇨ Параметры ⇨ Международные**:



6. Особенности интерфейса Microsoft Excel

После запуска Microsoft Excel на Рабочем столе Windows появляется основное окно этого приложения (рисунок 7.1).



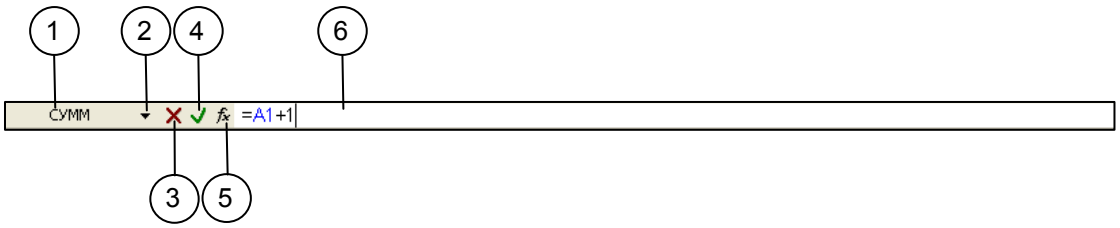
1. Заголовок окна приложения
2. Основное меню приложения
3. Панели инструментов **Стандартная** и **Форматирование**
4. Строка (панель) формул
5. Окно активного документа
6. Окно неактивного документа
7. Панель задач **Создание книги**
8. Поле редактирования со списком **Задать вопрос**
9. Помощник по офису
10. Панель задач Microsoft Windows
11. Строка состояния
12. Кнопки документов запущенных приложений на Панели задач Microsoft Windows
13. Контекстное меню Строки состояния Microsoft Excel
14. Панель инструментов **Быстрый запуск**
15. Свернутое окно документа
16. Режим работы Microsoft Excel
17. Кнопки прокрутки ярлычков
18. Ярлычок активного рабочего листа
19. Вешки разбивки
20. Полосы прокрутки
21. Заголовки строк
22. Заголовки столбцов
23. Активная ячейка
24. Границы активной ячейки
25. Маркер автозаполнения
26. Кнопка **Выделить весь лист**

Рисунок 7.1 – Многодокументный интерфейс Microsoft Excel

По умолчанию в нем открывается единственное окно документа, в котором отображается вновь созданная рабочая **Книга1** с тремя рабочими листами – **Лист1**, **Лист2** и **Лист3**. Основное окно Microsoft Excel является типичным окном многодокументного приложения (MDI – Multiple Document Interface). В нем имеется заголовок (1), основное меню (2), панели инструментов (3), дочерние окна открытых документов (5, 6, 15) и т. д. – т. е. все, что необходимо для одновременной работы с несколькими документами. Особенностью же этого окна является наличие в нем двух элементов управления:

1. *строки (панели) формул* и
2. *контекстного меню Строки состояния*.

Строка формул предназначена для ускорения и облегчения ввода и редактирования формул.



В ней, слева направо, присутствуют следующие компоненты:

1. Раскрывающийся список с полем редактирования **Имя**, который может выполнять несколько функций:
   а) В режиме **Готово** здесь отображается адрес текущей ячейки.
   б) В нем можно непосредственно ввести адрес нужной ячейки, и по нажатию клавиш **Enter** выделить ее.
   в) В режиме **Укажите** (выделения) в нем отображается количество помеченных ячеек, например, 2R × 2C для области 2 × 2 ячеек.
   г) Здесь можно присвоить *имя* любому выделенному блоку ячеек.
   д) В режимах **Ввод** и **Правка** здесь отображается меню функций, с помощью которого вызывается Мастер функций Excel.

2. Кнопка со стрелкой справа от списка **Имя** позволяет раскрыть перечень поименованных объектов, если они, конечно, имеются. Щелчок по нужному имени вызывает перемещение «фокуса ввода» на требуемый объект.
3. Кнопка **Отмена** аннулирует изменения аналогично клавише **Esc**. В ячейку возвращается ее прежнее содержание, и происходит возврат в режим **Готово**.
4. Кнопка **Ввод** предназначена для подтверждения сделанных изменений аналогично клавише **Enter**. При щелчке по ней происходит выход из режима **Ввод** или **Правка** и возврат в режим **Готово**, а так же присваивание введенного значения ячейке.

   Кнопки ***Отмена*** *и* ***Ввод*** *доступны только в режимах* ***Ввод*** *и* ***Правка***.
5. Кнопка **Вставка функции** служит для вызова **Мастера функций**, облегчающего ввод функций.
6. **Поле ввода** служит, собственно, для ввода и редактирования значений ячеек.

Контекстное меню Строки состояния активизируется по щелчку правой кнопкой мыши по ней. С его помощью для выделенного диапазона ячеек можно быстро подсчитать такие параметры, как их сумма, среднее значение, максимум, минимум и т.д. Результат такого мини-вычисления также отображается в строке состояния.

7. Режимы работы

Microsoft Excel может находиться в одном из следующих *режимов*, который отображается в левой части строки состояния:

1. **Готово** – в этом состоянии возможно перемещение по таблице и доступ ко всем элементам управления.
2. **Ввод** – режим ввода данных в клетку. Большая часть меню не доступна – доступны только функции, связанные со вводом данных.

3. **Правка** – режим редактирования ранее введенных данных. Очень похож на режим **Ввод**.
4. **Укажите** – режим выбора объекта (ячейки, диапазона и т.д.) для включения его в выражение. При этом выделяемый объект ограничивается бегущей пунктирной линией.

 *В различных режимах одни и те же функциональные клавиши могут действовать не одинаковым образом*.

8. Завершение работы

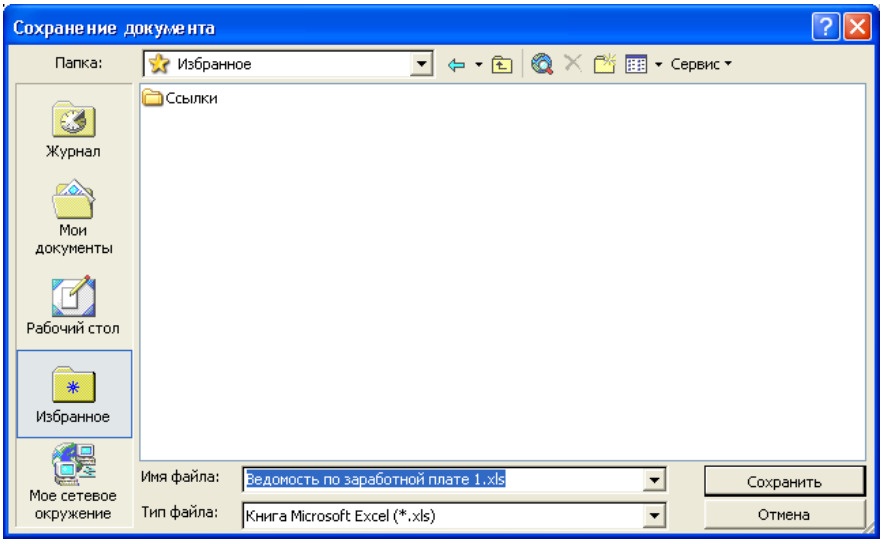
Хорошим тоном при работе с любым приложением, и Microsoft Excel в том числе, является сохранение вновь созданного документа с присвоением ему содержательного имени вместо заданного по умолчанию такого, например, как **Книга1**. Это, помимо присвоения документу осмысленного имени, позволяет также сразу иметь документ в виде дискового файла. Для сохранения документа в виде дискового файла необходимо осуществить такую последовательность шагов:

1.1. Выполнить команду **Файл ⇨ Сохранить**, или

1.2. щелкнуть по кнопке  **Сохранить** на панели инструментов **Стандартная**, или

1.3. нажать сочетание клавиш **Shift** + **F12**.

2. В раскрывшемся в результате окне диалога **Сохранение документа** указать:
   а) С помощью списка **Папка** – папку, в которой будет сохранена рабочая книга.
   б) В поле редактирования со списком **Имя файла** – новое имя книги.
   в) Нажать кнопку **Сохранить**.

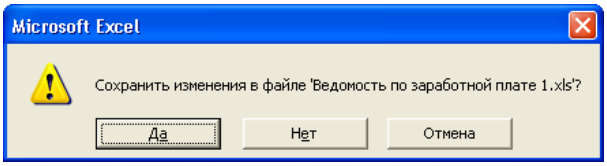


В дальнейшем, по мере ввода новой информации в рабочую книгу, ее необходимо периодически сохранять, выполняя ту же команду **Файл ⇨ Сохранить**, или щелкая по кнопке **Сохранить** на панели инструментов **Стандартная**, или нажимая сочетание клавиш **Shift** + **F12**. При этом окно диалога **Сохранение документа** больше не открывается – рабочая книга под уже имеющимся именем сохраняется «молча». Это окно диалога откроется вновь при сохранении существующей рабочей книги под новым именем, для чего необходимо выполнить команду **Файл ⇨ Сохранить как**.

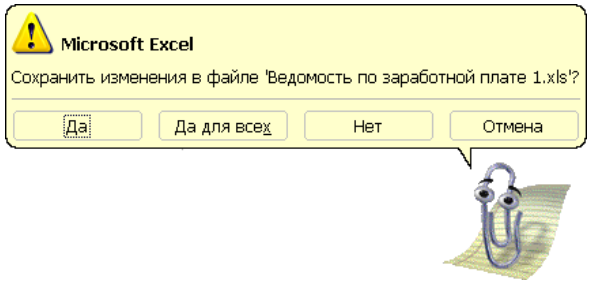
Для завершения работы Microsoft Excel необходимо:

- выбрать команду **Файл ⇨ Выход**, или
- нажать сочетание клавиш **Alt** + **F4**, или
- щелкнуть на кнопке **Закрыть** в строке заголовка основного окна приложения – если был открыт только один документ. Если же документов было открыто несколько – будет закрыт только текущий документ, а Microsoft Excel продолжит работу с другими открытыми документами.

Щелчок на кнопке **Закрыть окно** на панели меню активной рабочей книги приводит к закрытию только текущего документа, без завершения работы всего приложения. Если при этом небыли сохранены последние внесенные в рабочие книгу изменения, Excel выдаст соответствующее предупреждение для каждого из таких документов.



Для сохранения изменений при закрытии рабочей книги следует щелкнуть на кнопке **Да**. При включенном режиме отображения *Помощника* по офису (англ. *Office Assistant*) подобного рода сообщение будет отображаться Помощником.



ЛЕКЦИЯ № 8

ПРОГРАММНЫЕ СРЕДСТВА РАБОТЫ С БАЗАМИ И ХРАНИЛИЩАМИ ДАННЫХ

План

1. Технология баз данных
2. История развития БД
3. Классификация БД
4. Модели данных
5. Реляционная БД

5.1. Основные положения

5.2. Ключи: первичные и внешние

5.3. Нормализация и денормализация БД

5.4. Язык SQL

6. Постреляционные модели данных

6.1. Объектно-ориентированные БД

6.2. Гибридные БД

7. СУБД и сети

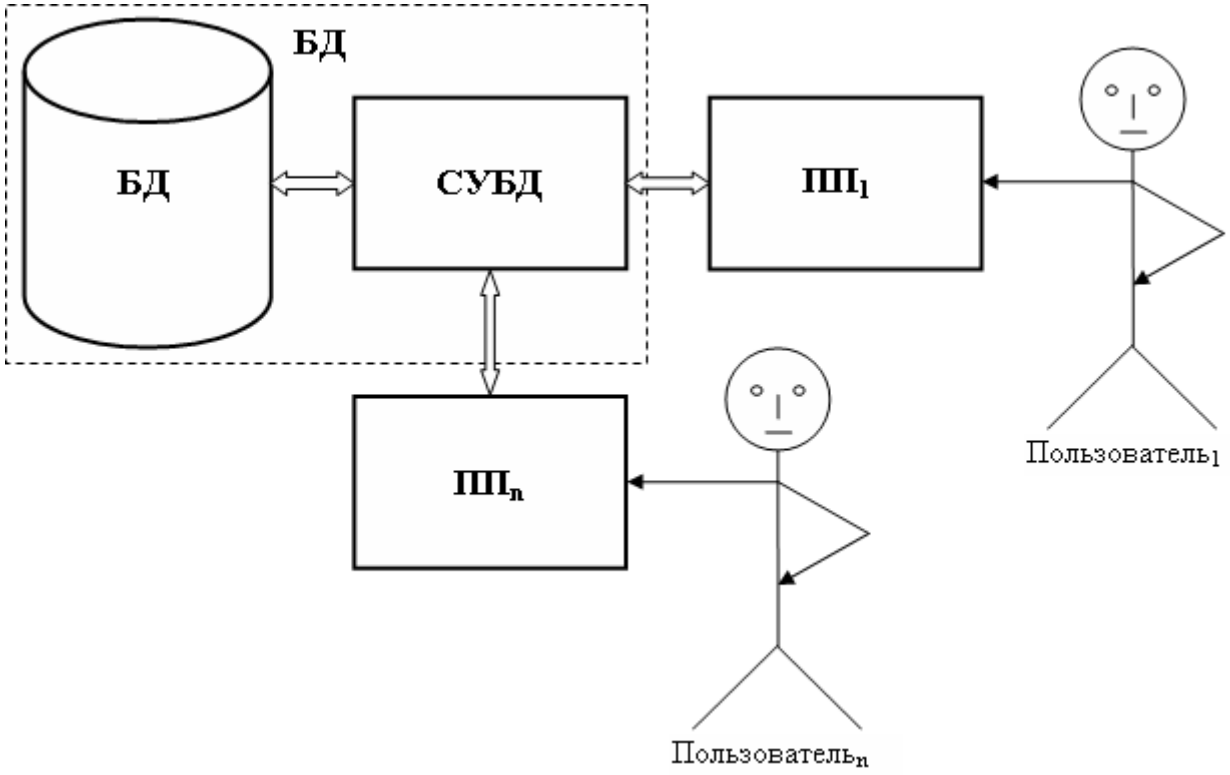
1. Технология баз данных

Базой данных (*БД*) называется совокупность структурированной информации, относящейся к одной предметной области, которая обеспечивает централизованное накопление и коллективное многоцелевое использование различными приложениями. Совокупность программ, реализующих функции доступа к БД, называется *системой управления базой данных* (*СУБД*). Программы, производящие специфическую обработку данных в БД, называются *прикладными программами* (*ПП*).

Хранение информации в БД подчиняются некоторым общим принципам, среди которых в первую очередь следует выделить:

- *целостность и непротиворечивость данных*, под которыми понимается как физическая сохранность данных, так и предотвращение неверного использования данных, поддержка допустимых сочетаний их значений, защита от структурных искажений и несанкционированного доступа;
- *минимальная избыточность данных*, которая обозначает, что любой элемент данных должен храниться в базе в единственном экземпляре, что позволяет избежать необходимости дублирования операций.

Схема организации работы с БД в самом общем виде представлена на рисунке 8.1.



БД – база данных
СУБД – система управления базой данных
$ПП_1$ – прикладная программа$_1$
$ПП_n$ – прикладная программа$_n$

Рисунок 8.1 – Схема организации работы с БД

Согласно этой схеме, в системе имеется база данных, в которой хранится некоторая информация. Физически база данных представляет собой набор взаимосвязанных файлов на внешних запоминающих устройствах компьютера. То есть, база данных – это просто файловое хранилище информации, и не более. Сама по себе БД не представляет никакого интереса, если бы не было СУБД. СУБД берет на себя все операции по управлению БД. Все СУБД друг от

друга отличаются лишь способом организации работы с файловой системой. То есть, каждая СУБД поддерживает работу только с одной *моделью данных*. Иногда, для краткости, СУБД называют просто базой данных. Но всегда, когда говорят БД, речь идет именно о СУБД.

Взаимодействие прикладных программ пользователя с БД представляет собой 3-х уровневую модель:

- Первый уровень – это прикладные программы ($ПП_1 \div ПП_n$ на рисунке 8.1), которые решают целевые задачи пользователя. Для получения необходимой информации они обращаются к СУБД на специальных языках запросов. Прикладные программы «не знают» где и как хранится информация.
- Второй уровень – это СУБД. Она «знает» где и как храниться информация. И, по запросам прикладных программ, обеспечивает их необходимой информацией.
- Третий уровень – БД – это просто «хранилище» информации.

В этой модели между прикладными программами, которым необходима информация, и файловой системой компьютера расположена СУБД – она управляет всеми операциями обмена информацией между ними. Поскольку программное обеспечение СУБД функционально располагается между системным и приложениями пользователя, его относят к разряду *промежуточного программного обеспечения* (англ. *middleware*).

Одним из важнейших назначений СУБД является обеспечение *независимости данных*. Под этим термином понимается независимость данных и использующих их прикладных программ друг от друга в том смысле, что изменение одних не приводит к изменению других.

Технология баз данных, в самом общем виде, решает две основные задачи:

1. длительное хранение и обработку огромных массивов информации и
2. одновременный и независимый доступ к ней множества прикладных программ, написанных, на различных языках программирования.

2. История развития БД

СУБД проделали длительный путь эволюции от:

1. *систем управления файлами* через
2. *иерархические* и
3. *сетевые* базы данных к
4. *реляционным*,
5. *обекно-ориентированным* и
6. *гибридным* моделям.

Задача длительного хранения и обработки информации появилась практически сразу после создания первых компьютеров. Данные записываются на жесткие диски компьютера в виде файлов определенной структуры, но связь между данными этих файлов отсутствует. Строго говоря, *системы управления файлами* нельзя классифицировать как СУБД, так как обычно они являются частью операционной систем и ничего не знают о внутреннем содержимом файлов. Это знание заложено в прикладных программах, работающих с файлами.

Отдельные файлы в подобных системах не имеют единого интерфейса, и каждая программа трактует его содержание по-своему. Поэтому для каждого типа файлов необходимо иметь набор программ, манипулирующих такими файлами, включая чтение, запись и обновление данных. Создание программ, обрабатывающих обычные («плоские») файлы, осуществляется, как правило, на языках программирования высокого уровня (таких как C, C++, Pascal и т. д.) и требует высоких профессиональных навыков программиста, так как ему нужно продумать не только логическую, но и физическую структуру хранения данных. Это приводит к тому, что между приложением и файлом образуется тесная связь. Вся информация о структуре файла закодирована в приложении. Другое приложение, обращающееся к тому же файлу, вынуждено дублировать уже существующий код. По мере увеличения числа приложений растет сложность управления такой базой данных. Изменения схемы данных приводят к необходимости изменения каждого программного компонента, для которого

это актуально. То есть любое изменение структуры данных вызывает каскадные изменения программ, зависящих от этой структуры. Например, поле, содержавшее шестизначный почтовый индекс, необходимо расширить до 9-и знаков. При этом, во-первых, существующие данные придется преобразовать в новый формат с помощью специальной программы. А, во-вторых, потребуется переписать все программы, работающие с файлом этого формата, что приводит к немалым затратам времени и усилий.

Для решения этой задачи в конце 60-х годов XX века были разработаны специализированные программы, получившие в дальнейшем название *системы управления базами данных* (*СУБД*). Исторически первые базы данных создавались на основе файловых систем, хранящих в себе «сырые» (неупорядоченные) данные. А СУБД и были разработаны как раз с целью решения проблем, возникающих при создании специализированных прикладных программ обработки данных.

В базе данных информация представлена единообразно. Сведения о том, как структурированы данные и как они связаны друг с другом, хранятся в самой базе в виде *метаданных*. Поэтому СУБД более гибкие в использовании. Они позволяют приложениям работать с данными на логическом уровне, игнорируя их физическую структуру. Изменение формата данных может не вызвать никаких изменений в интерфейсе взаимодействия с ними, а следовательно – в самих приложениях. Основное преимущество баз данных – это наличие унифицированного интерфейса. Не нужно «изобретать колесо» и постоянно создавать новые модули манипулирования данными. Все обращения к базе данных реализуются через специальный модуль СУБД – сервер БД. Он «понимает» специальный язык запросов, при помощи которого описываются все манипуляции с содержанием БД. Наличие подобного языка запросов значительно упрощает создание процедур обмена информацией между прикладными программами и БД. Программист, работающий с базой данных, не заботится о том, как эти данные хранятся, и приложения, взаимодействующие с СУБД, «не знают» о способе записи данных на диск.

«Снаружи» виден лишь логический образ данных, и это позволяет менять код СУБД, не затрагивая код самих приложений. При этом операции можно группировать в так называемые транзакции, выполняемые по принципу «все или ничего».

Первая промышленная СУБД была разработана в 1968 году компанией Rockwell в сотрудничестве с IBM. Названная *IMS* (Information Management System – система управления информацией), она заложила основу концепции современных СУБД. Ключевым новшеством IMS было разделение данных и функций деловой логики. Прикладные программисты получили возможность работать с информацией на логическом уровне, а база данных брала на себя задачу физического хранения. Подобное «разделение труда» привело к резкому скачку производительности. Еще одним изобретением стал язык DL/1 (Data Language/1) Это был специализированный язык составления *нерегламентированных* (англ. *ad hoc*) запросов к базе данных. Его появление сделало ненужным дорогостоящее программирование на таких языках, как COBOL и FORTRAN, популярных в то время. В СУБД IMS, применяемой до сих пор, реализована иерархическая модель данных, в которой существует один-единственный путь от корня иерархии к каждой записи. Такая модель данных стала основой для других систем управления данными, и она же дала толчок к следующему витку изобретений, в частности – *сетевой модели данных*.

Сетевой подход к организации данных является расширением иерархического. В сетевой модели любая запись может участвовать в нескольких отношениях «предок-потомок». Это позволяло обходить целый ряд ограничений иерархической модели. Концепция сетевой модели данных связана с именем Чарльза Бахмана, который в середине 70-х годов руководил проектом IDMS (Integrated Data Management System – интегрированная система управления данными) в компании General Electric. Он же изобрел и «диаграммы Бахмана» описывающие сетевые базы данных. За свой труд в 1973 году Ч. Бахман получил награду Тьюринга.

А тем временем научный сотрудник компании IBM доктор Эдгар Кодд (Edgar Codd) работал над эпохальным документом для Ассоциации производителей вычислительной техники (Association for Computing Machinery, ACM). В июне 1970 года этот документ был опубликован в ACM Journal под названием «Реляционная модель для больших банков совместно используемых данных» («A Relational Model of Data for Large Shared Data Banks»). Он в корне изменил теорию баз данных и принес доктору Кодду награду Тьюринга в 1981 году.

Доктор Кодд предложил реляционную модель, в которой данные можно было свободно описывать в их естественном виде без каких-либо ограничений, накладываемых средой физического хранения. Следовательно, это позволяло создать язык высокого уровня, способный работать с данными независимо от того, как они хранятся в компьютере.

Практическим результатом работы доктора Кодда явилось создание двух СУБД: System R компании IBM и Ingres Калифорнийского университета Беркли. В обеих была реализована реляционная модель данных и *язык структурированных запросов*. Последний в СУБД System R первоначально назывался SEQUEL (Structured English Query Language – структурированный английский язык запросов). Позднее у него появилось другое название – SQL (Structured Query Language). Одни люди эту аббревиатуру произносят как «сиквел», другие – как «эскюэль».

3. Классификация БД

Базы данных можно классифицировать по множеству самых разнообразных, часто пересекающихся, признаков. Наиболее широко распространенными являются классификации по:

- *типу принятой модели данных*,
- *технологии обработки данных*,
- *способу доступа*,
- *скорости обработки информации*,

- *области применения* и
- *количеству пользователей*.

По типу принятой модели данных БД бывают:

- *иерархические*,
- *сетевые*,
- *реляционные*,
- *объектно-ориентированные* и
- *гибридные* или *объектно-реляционные*.

Иерархические базы данных основаны на иерархической модели данных, в которой связь между объектами базы данных образует перевернутое дерево. При такой модели каждый нижележащий элемент иерархии соединен только с одним расположенным выше элементом.

Сетевые базы данных основаны на сетевой модели данных. В которой связи между объектами могут быть установлены в произвольном порядке.

Реляционные базы данных основаны на реляционной модели данных, в которой каждая единица данных в базе данных однозначно определяется именем таблицы (называемой отношением), идентификатором записи (кортежа) и именем поля.

Объектно-ориентированные базы данных определяют как новое поколение баз данных, основанное на сочетании трех принципов: реляционной модели, стандартов на описание объектов и принципов объектно-ориентированного программирования.

Объектно-редяционные базы данных содержат объектно-ориентированные механизмы построения структур данных (как минимум, механизм наследования и поддержки методов) в виде расширений языка и программных надстроек над ядром СУБД.

В зависимости от технологии обработки данных БД бывают:

- *централизованные* и
- *распределенные*.

В *централизованных* БД все объекты хранится целиком на одном компьютере. Если компьютер входит в состав сети, то возможен доступ к этой БД других компьютеров.

Распределенная БД состоит из нескольких, возможно пересекающихся или дублирующих друг друга БД, которые могут находиться на различных компьютерах, объединенных в сеть.

По способу доступа БД делятся на:

- *локальные*,
- *удаленные* (*сетевые*) и
- *встраиваемые*.

Локальный доступ предполагает, что БД и СУБД находятся на одном и том же компьютере.

Удаленный доступ – это обращение к БД, которая хранится на одном из компьютеров, входящих в компьютерную сеть. Удаленный доступ, в свою очередь, может быть выполнен по принципу:

- *файл-сервер* или
- *клиент-сервер*.

Концепция *файл-сервера* предполагает наличие компьютера, выделенного под файловый сервер, на котором централизованно хранятся файлы БД, а копии СУБД находится на каждом клиентском компьютере. Для этой архитектуры характерен коллективный доступ к общей базе данных на файловом сервере. Запрошенные данные транспортируются с файлового сервера на рабочие станции, где их обработка выполняется средствами локальных СУБД. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Однако при большой интенсивности запросов к централизованной БД увеличивается нагрузка на каналы связи, что приводит к снижению общей производительности системы.

Архитектура *клиент-сервер* предполагает, что сервер, выделенный для хранения централизованной БД, дополнительно производит обработку клиентских запросов. Запрос на обработку данных выдается клиентом и

передается по сети на сервер баз данных. Обработанные данные транспортируются по сети от сервера к клиенту. То есть, клиент – программа, которой понадобились данные из БД, посылает запрос серверу – программе, управляющей ведением БД, на специальном языке запросов. Сервер пересылает программе данные, являющиеся результатом поиска в БД по ее запросу. Этот способ удобен тем, что программа-клиент не обязана содержать все функции поддержания и ведения БД – этим занимается сервер. В результате упрощается написание программ-клиентов. Кроме того, к серверу может обращаться любое количество различных программ. Недостаток клиент-серверных СУБД состоит в самом факте существования сервера (что плохо для локальных программ – в них удобнее *встраивать* СУБД) и больших вычислительных ресурсах, потребляемых сервером.

Учитывая возросшую мощность компьютеров, а так же широкое распространение БД в самых различных областях, в последнее время архитектура клиент-сервер стала применяться и на одиночных вычислительных системах. То есть, и программа-клиент и программа-сервер находятся на одном и том же компьютере. Следующим шагом в этом направлении является объединение программы-клиента и программы-сервера в одну программу. Такое решение называется *встраиванием* СУБД в клиентскую программу. Встраиваемая СУБД представляет собой программную библиотеку, которая позволяет унифицированным образом хранить большие объемы данных на локальной машине. Доступ к данным может происходить посредством запросов, либо путем вызова функций библиотеки из приложения пользователя. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют развертывания сервера.

По скорости обработки информации БД делятся на:

- *операционные* (*operational*) или *рабочие* (*production*) и
- *хранилища данных* и *многомерные хранилища данных* (*data warehouse*, *OLAP*)

Операционные (*operational*), или *рабочие* (*production*), базы данных обладают высокими скоростями реакции на запрос извлечения и представления информации.

Хранилища данных и *многомерные хранилища данных* (*data warehouse*, *OLAP*) – это базы данных с очень большими объемами информации, подготовка представления которой занимает значительный объем времени.

В зависимости от области применения все БД можно разделить на:

- *профессиональные* (или *промышленные*) и
- *персональные* (или *настольные*).

Профессиональные СУБД представляют собой программную основу для разработки автоматизированных систем управления крупными экономическими объектами. На их основе создаются комплексы управления и обработки информации крупных предприятий, банков или даже целых отраслей.

Основными условиями, которым должны удовлетворять профессиональные СУБД, являются:

- возможность организации совместной параллельной работы большого количества пользователей;
- масштабируемость, то есть возможность роста системы пропорционально расширению управляемого объекта;
- переносимость на различные аппаратные и программные платформы;
- устойчивость по отношению к различного рода сбоям.

Наиболее известными в настоящее время профессиональными СУБД являются Oracle и MySQL.

Персональные системы управления данными – это программное обеспечение, ориентированное на решение задач локального пользователя или компактной группы пользователей и предназначенное для использования на персональных компьютерах. Это объясняет и их второе название – настольные. Определяющими характеристиками настольных СУБД являются:

- относительная простота эксплуатации, позволяющая создавать на их основе работоспособные приложения как «продвинутым» пользователям, так и тем, чья квалификация не особенно высока;
- относительно ограниченные требования к аппаратным ресурсам.

Исторически первой среди персональных СУБД, получивших массовое распространение, стала dBASE фирмы Ashton-Tate. В настоящее время наиболее известными СУБД этого класса являются проприетарная Access для ОС Microsoft Windows и свободная OpenOffice Base для ОС Linux.

Необходимо также отметить, что в последние годы наметилась устойчивая тенденция к стиранию четких граней между настольными и профессиональными системами. Последнее, в первую очередь, объясняется тем, что разработчики в стремлении максимально расширить потенциальный рынок для своих продуктов постоянно расширяют набор их функциональных характеристик. Среди СУБД, которые, условно говоря, занимают промежуточное положение между настольными и промышленными системами, можно назвать Microsoft SQL Server.

По количеству одновременно работающих пользователей БД бывают:

- *однопользовательские* и
- *многопользовательские*.

Однопользовательские БД размещаются на локальных компьютерах отдельных пользователей. Как правило – это персональные базы данных.

Многопользовательские БД предоставляют одновременный доступ к информации большому количеству пользователей. В этом случае каждый пользователь со своего компьютера получает доступ к общей для всех централизованной базе данных.

4. Модели данных

Центральным понятием в области баз данных является понятие модели данных. Она является ключевым понятием любой БД. Под моделью представления данных понимают набор принципов, определяющих

организацию логической структуры хранения данных и совокупность операций над этими структурами. В теории систем управления базами данных выделяют модели трех основных типов:

1. *иерархическую*,
2. *сетевую* и
3. *реляционную*.

Первые две из них основаны на графовом представлении информации об объектах, последняя – на табличном.

Самыми первыми промышленными базами данных были иерархические. Они основаны на древовидной структуре хранения информации и в этом смысле напоминает файловую систему компьютера. С точки зрения организации хранения информации база данных состоит из упорядоченного набора деревьев одного типа, т. е. каждая запись в базе данных реализована в виде отношения «предок-потомок». На рисунке 8.2 приведена схема организации такой иерархической базы данных на примере воинской части.

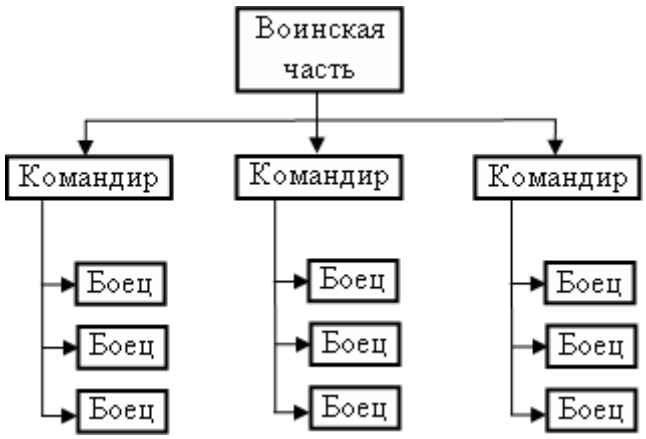


Рисунок 8.2 – Схема организации иерархической БД

В ней объекты «Командир» являются потомками для объекта «Воинская часть», а объекты «Боец» являются потомками для объекта «Командир». Вся база представляет собой иерархическое дерево, используя которое можно ответить на вопросы: сколько бойцов в части, сколько бойцов у конкретного командира и т. п.

Иерархические базы данных наиболее пригодны для моделирования структур, являющихся иерархическими по своей природе. Если при

моделировании воинской части это условие легко выполняется, поскольку воинские подразделения подразумевают единоначалие, то в остальных сферах деятельности разработчики сталкиваются с тем, что иерархические базы данных плохо справляются с моделированием. Они не позволяют организовать более сложные отношения, т. к. иерархия подразумевает наличие только одного родителя.

К основным недостаткам иерархической модели можно отнести следующие:

- сложность отображения связей «многие ко многим»;
- иерархия в значительной степени усложняет операции включения информации о новых объектах в базу данных и удаления устаревшей;
- доступ к любому узлу возможен только через корневой.

Иерархические базы данных в силу своих недостатков просуществовали не долго и были потеснены сетевыми базами данных, являющимися, по сути, расширением иерархических, т. е. иерархическая модель данных является частным случаем сетевой. Если в иерархических базах данных структура «запись-потомок» должна иметь только одного предка, то в сетевой базе данных потомок может иметь любое количество предков. Термином «сетевая» обозначается модель связей в базе данных, когда каждая запись может находиться в отношениях «многие ко многим» с другими записями.

Например, если создать каталог книг, то одна и та же книга может относиться сразу к нескольким разделам, и в этом случае отношения будут выглядеть так, как это представлено на рисунке 8.3.

Еще одним примером сетевой модели данных может служить организация сети железнодорожных путей. В ней каждая крупная железнодорожная станция может иметь связи (пути сообщения) с несколькими другими станциями (связь «многие ко многим»). Примерно таким же образом выглядит графическое представление любой сетевой базы данных.

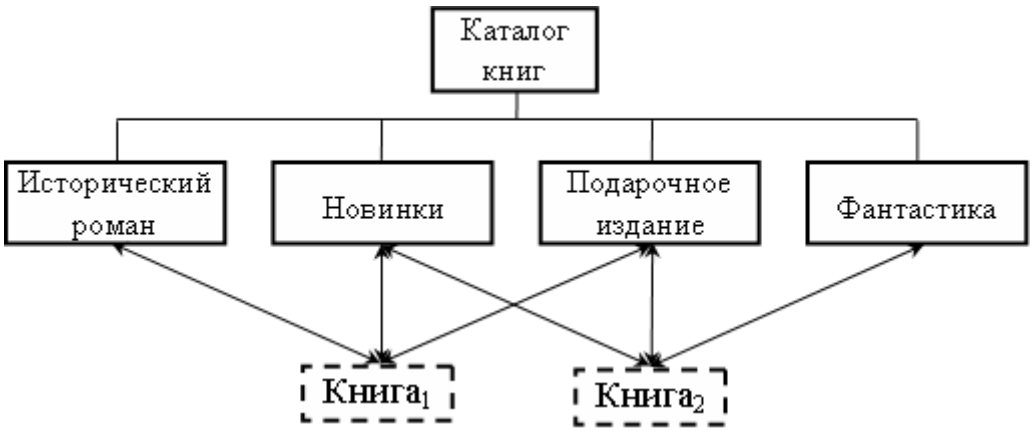


Рисунок 8.3 – Схема организации сетевой БД

Основной недостаток сетевой модели состоит в ее сложности. Другим недостатком является возможная потеря данных при реорганизации базы данных. Кроме того, в сетевой модели данных представление, используемое прикладной программой, сложнее, чем в иерархической модели, поэтому процесс создания прикладных программ может оказаться сложнее.

5. Реляционная БД

5.1. Основные положения

С конца 80-х годов XX века доминирующими стали системы управления реляционными базами данных. С этого времени такие СУБД стали стандартом de-facto. Основным положительным их отличие от иерархической и сетевой моделей – отсутствие жестких связей между данными. В ней нет явных указателей на предков и потомков, а все данные представлены в виде простых таблиц, состоящих из строк и столбцов, на пересечении которых расположены конкретные значения. При этом столбцы определяют структуру таблицы, а строки – число записей в ней. Как правило, одна база данных содержит несколько таблиц, которые могут быть, как связаны друг с другом, так и независимы друг от друга. Все операции над данными в реляционной БД сводятся к операциям над таблицами.

На рисунке 8.4 показана одна из таблиц базы данных, в которой приведено деление на категории компьютерных комплектующих поставляемых

компанией. Каждая строка этой таблицы представляет собой одну категорию товарных позиций, для описания которой используется два поля:

- id_category – уникальный номер категории и
- name – название категории товаров.

Таблица categories	
id_category	name
1	Процессоры
2	Материнские платы
3	Видеоадаптеры
4	Жесткие диски
5	Оперативная память

Рисунок 8.4 – Таблица реляционной БД

В строках таблиц реляционной БД часть полей содержит данные, относящиеся непосредственно к записи, а часть – ссылки на записи других таблиц. Таким способом устанавливаются связи между записями в различных таблицах реляционной модели данных. Именно благодаря наличию связей такая форма организации информации получила название *реляционной* (от англ. *relation* – отношение).

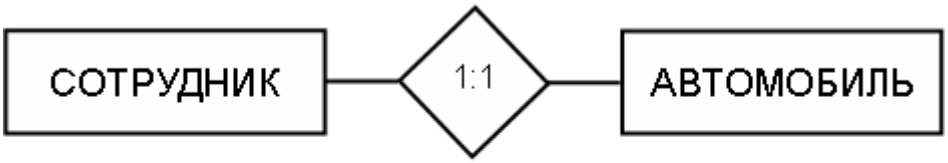
На рисунке 8.5 приведена структура базы данных, состоящей из двух таблиц, categories и products. Таблица categories определяет число и название категорий товара, а таблица products содержит описание самих товарных позиций. Одной товарной позиции соответствует одна строка таблицы products . Последнее поле этой таблицы содержит значения из поля id_category таблицы categories. По этому значению можно однозначно определить, к какой категории относится данная товарная позиция. Таким образом, таблицы получаются связанными друг с другом.

Таблица products		
id_product	…	id_category
1	…	2
2	…	5
3	…	1
4	…	1
5	…	1
6	…	5
7	…	2
8	…	4
9	…	3

Таблица categories	
id_category	name
1	Процессоры
2	Материнские платы
3	Видеоадаптеры
4	Жесткие диски
5	Оперативная память

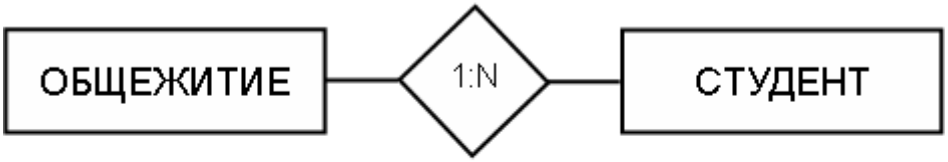
Рисунок 8.5 – Связанные таблицы БД

Строки таблиц реляционной БД могут быть связаны друг с другом одним из трех способов. Простейшее отношение – «один к одному». В этом случае строка первой таблицы соответствует одной-единственной строке второй таблицы. На диаграммах такое отношение выражается записью 1:1, например:



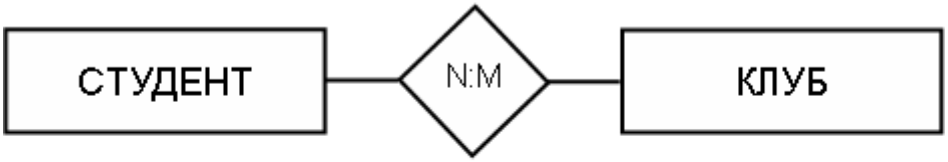
СЛУЖЕБНЫЙ АВТОМОБИЛЬ

Отношение «один ко многим» означает ситуацию, когда строка одной таблицы соответствует нескольким строкам другой таблицы. Это наиболее распространенный тип отношений. На диаграммах он выражается записью 1:N, например:



ОБЩЕЖИТИЕ-СТУДЕНТ

Наконец, при отношении «многие ко многим» строки первой таблицы могут быть связаны с произвольным числом строк во второй таблице. Такое отношение записывается как N:M, например:



СТУДЕНТ-КЛУБ

Такие отношения в реляционной БД реализуются при помощи дополнительных (вспомогательных) таблиц.

Таблицы реляционной БД имеют свои особенностями, которые заключаются в следующем:

- Данные хранятся в таблицах, состоящих из строк (*записей*, *кортежей*, англ. *records*) и столбцов (*полей*, *атрибутов*, англ. *fields*).
- На пересечении каждого столбца и строки находится *только одно неделимое* (*атомарное*) *значение*.
- У каждого столбца есть свое имя, которое служит его названием, и все значения в одном столбце имеют один тип. Например (рисунок 8.4), в столбце id_category все значения имеют целочисленный тип, а в столбце name – текстовый.
- Столбцы располагаются в определенном порядке, который задается при создании таблицы, в отличие от строк, которые располагаются в произвольном порядке. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.
- Запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объекты запросов.

Сила реляционных баз данных заключается не только в уникальной организации информации в виде таблиц. Запросы к таблицам базы данных сами возвращают таблицы, которые называют результирующими таблицами. Даже если возвращается одно значение, это все равно считается таблицей, состоящей

из одного столбца и одной строки. То, что запрос возвращает таблицу очень важно. Это означает, что результаты запроса можно записать обратно в базу данных в виде таблицы, а результаты двух или более запросов, которые (т. е. результаты) имеют одинаковую структуру, можно объединить в одну таблицу. И, наконец, это говорит о том, что результаты запроса сами могут стать предметом дальнейших запросов.

5.2. Ключи: первичные и внешние

Строки в реляционной БД неупорядочены, т. е. в таблице нет «первой», «последней», «десятой» или «сорок первой» строки, а есть просто неупорядоченный набор записей. Возникает вопрос: каким же образом выбрать в таблице конкретную запись? Для этого в правильно спроектированной базе данных для каждой таблицы создается один или несколько столбцов, значения которых во всех записях различны. Такой столбец называется *первичным ключом* (англ. *primary key – PK*). Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа, т. е. благодаря наличию первичных ключей каждая строка таблицы обладает своим уникальным идентификатором. Так на рисунке 8.5 в качестве первичного ключа таблицы products выступает поле id_product, а в качестве первичного ключа таблицы categories – поле id_category. Если ключ записи состоит из значений нескольких полей, то он называется *составным*, а если из одного атрибута – *простым*.

По способу задания первичных ключей различают:

- *логические* (они же *естественные*) и
- *суррогатные* (они же *искусственные*).

Для логического задания первичного ключа необходимо выбрать в базе данных то, что естественным образом определяет запись. Например, в таблице базы данных, содержащей паспортные данные жителей, уникальный индекс можно создать для поля «Номер паспорта», поскольку каждый такой номер является единственным в своем роде. Однако дата рождения уже не уникальна,

поэтому поле «Дата рождения» не может выступать в качестве первичного ключа.

Если подходящих полей для естественного задания первичного ключа не находится, пользуются суррогатным ключом. Он представляет собой дополнительное поле в базе данных, предназначенное для обеспечения записей первичным ключом (именно такой подход принят в приме на рисунке 8.5).

⚠ *Даже если в БД содержится естественный ключ, лучше использовать суррогатный ключ, поскольку их применение позволяет абстрагировать первичный ключ от реальных данных. В этом случае облегчается работа с таблицами, поскольку суррогатные ключи не связаны ни с какими фактическими данными этой таблицы.*

Одним из предназначений первичных ключей является однозначная организация логических связей разных таблиц друг с другом. Хотя, формально, таблицы и не обязаны содержать первичные ключи, но это очень желательно, если данные связаны друг с другом. Например (рисунок 8.5), столбец id_category таблицы products принимает значения из столбца id_category таблицы categories и называется он *внешним* или *вторичным* ключом (англ. *foreign key – FK*). Благодаря такой связи можно выстроить иерархию товарных позиций и определить, к какой категории относятся конкретный товар.

5.3. Нормализация и денормализация БД

Нормализацией схемы базы данных называется процедура, производимая над базой данных с целью удаления в ней избыточности и непротиворечивости. Например, в рассматриваемом ранее магазине компьютерных комплектующих необходимо учитывать не только товарные позиции, но и покупки. Это может быть полезно для автоматического пополнения запасов на складе, ведения бухгалтерского учета, отслеживания предпочтений покупателей или для предоставления им скидок. Для учета сделок можно использовать таблицу orders, структура которой представлена на рисунке 8.6.

Таблица orders				
id_order	client	time	number	id_product
1	Иванов И.И.	2015-10-04	1	3
2	Петров П.П.	2015-02-10	2	2
3	Сидоров С.С.	2015-02-18	4	3
4	Иванов И.И.	2015-03-10	1	6
5	Иванов И.И.	2015-03-17	1	5

Рисунок 8.6 – Избыточность в таблице БД

Она содержит первичный ключ id_order, фамилию и инициалы покупателя client, время покупки time, число приобретенных товаров number и вторичный ключ id_product, по которому в таблице products можно определить приобретенную товарную позицию. Но эта таблица избыточна – для каждого заказа вводится имя покупателя, что может привести к ошибкам и замедлить операции, связанные с выбором заказов каждого из покупателей. Для нормализации этой таблицы ее необходимо разбить на две, как показано на рисунке 8.7.

Таблица orders				
id_order	id_client	time	number	id_product
1	1	2015-10-04	1	3
2	2	2015-02-10	2	2
3	2	2015-02-18	4	3
4	1	2015-03-10	1	6
5	1	2015-03-17	1	5

Таблица clients			
id_client	name	phone	email
1	Иванов И.И.	577-78-36	ivanov@rambler.ru
2	Петров П.П.	577-98-78	petrov@mail.ru
3	Сидоров С.С.	577-66-10	sidorov@ukr.net

Рисунок 8.7 – Нормализованная БД

Теперь в таблице orders вместо фамилии покупателя используется вторичный ключ id_client для связи с таблицей clients, в которую вынесена вся информация о каждом покупателе. Как видно из рисунка 8.7, разбиение таблицы orders на две связанные таблицы позволило добавить контактную информацию для каждого из покупателей. В противном случае потребовалось бы вводить дополнительные поля в таблицу orders и дублировать информацию при каждой новой покупке. При этом изменение, например, контактной информации потребовало бы обновления всех записей таблицы orders, соответствующих данному покупателю. Очевидно, что нормализация несет в себе немало преимуществ: в нормализованной базе данных уменьшается вероятность возникновения ошибок, она занимает меньше мечта на жестком диске и т. д.

⚠ *Хотя в теории баз данных и говорится о том, что схема базы данных должна быть полностью нормализована, в реальности при работе с полностью нормализованными базами данных необходимо применять весьма сложные запросы, что приводит к обратному эффекту – замедлению работы базы данных. Поэтому иногда для упрощения запросов даже прибегают к обратной процедуре –* ***денормализации***.

5.4. Язык SQL

На языке баз данных команды, обращающиеся к базе, называются *инструкциями* либо *запросами*. Язык, при помощи которого описываются такие запросы к реляционной базе данных называется SQL (Structured Query Language – язык структурированных запросов). Он является стандартом de-facto языка работы с реляционными базами данных.

Таким образом, для того чтобы получить какую-либо информацию из базы данных, необходимо направить базе данных запрос, созданный с использованием SQL, результатом выполнения которого будет результирующая таблица. SQL не является языком программирования, т. е. в отличие от языков программирования высокого уровня, таких как C, C++, Pascal и т. д. с его помощью невозможно создать полноценную программу. Все запросы

выполняются либо в специализированных программах, либо из прикладных программ при помощи специальных библиотек.

Несмотря на то, что SQL называется «языком запросов», он представляет собой нечто большее, чем просто инструмент для создания запросов. С помощью SQL осуществляется реализация всех возможностей, которые предоставляются пользователям разработчиками СУБД. В составе SQL могут быть выделены следующие группы инструкций:

- *определения данных* (англ. *DDL – Data Definition Language* – язык описания данных), которые позволяют манипулировать схемой базы данных, то есть создавать, изменять и удалять объекты базы данных;
- *манипулирования данными* (англ. *DML – Data Manipulation Language* – язык манипулирования данными), которые позволяют производить извлечение, добавление, изменение и удаление данных;
- *управления*. К ним относятся инструкции:
  - *назначения прав доступа* и
  - *управления транзакциями*.

  Под транзакцией понимают логически завершенную единицу работы, которая содержит одну или более элементарных операций обработки данных. Все действия, составляющие транзакцию, должны либо выполниться полностью, либо полностью не выполниться.

Не смотря на то, что язык запросов SQL строго стандартизован, существует множество его диалектов. По сути, каждая СУБД реализует собственный диалект со своими особенностями и ключевыми словами, недоступными в других. Такая ситуация связана с тем, что стандарт SQL появился достаточно поздно, в то время как компании-поставщики СУБД существуют давно и обслуживают большое число клиентов, для которых требуется обеспечить обратную совместимость со старыми версиями программного обеспечения.

6. Постреляционные модели данных

Теория баз данных не стоит на месте. На очередном витке развития появились новые технологии, расширяющие реляционную модель данных. Наиболее перспективные из них, это:

- *объектно-ориентированные* и
- *гибридные* или *объектно-реляционные* базы данных.

6.1. Объектно-ориентированные БД

Объектно-ориентированные базы данных (ООБД) появились в результате слияния объектно-ориентированного подхода в программировании и существующих теорий баз данных. Основной составляющей объектно-ориентированной модели данных является объект, который объединяет *состояние* (*данные*) и *поведение* (*методы* или *подпрограммы-функции*). Поддерживаются также *инкапсуляция*, *полиморфизм* и *наследование*.

Главным преимуществом ООБД является упрощение программного кода. Приложения получают данные в формате того языка программирования, на котором они написаны. Традиционная реляционная база данных возвращает значения всех полей в текстовом виде, а затем они приводятся к необходимым типам данных. В ООБД этот этап ликвидирован. Методы манипулирования данными всегда остаются одинаковыми независимо от того, находятся данные на диске или в памяти. При этом изменения, вносимые в объекты, оптимальным образом переносятся из памяти на диск.

С помощью ООБД решаются две проблемы. Во-первых, сложные информационные структуры выражаются в них лучше, чем в реляционных базах данных, а во-вторых, устраняется необходимость транслировать данные из формата, который поддерживается в СУБД во внутренний формат приложения. Например, в реляционной СУБД размерность целых чисел может составлять 11 цифр, а в используемом языке программирования – 16. В результате программисту необходимо учитывать эту ситуацию.

Объектно-ориентированные БД окупаются сполна, если отношения между данными очень сложны. В таком случае производительность ООБД

оказывается выше, чем у реляционных СУБД. Если же данные менее сложны, дополнительные функции оказываются избыточными.

Основным недостатком объектно-ориентированных баз данных является их тесная связь с применяемым языком программирования. К данным, хранящимся в реляционной СУБД, могут обращаться любые приложения, тогда как, например, Java-объект, помещенный в БД, будет представлять интерес лишь для приложений, написанных на Java.

6.2. Гибридные БД

Гибридные (объектно-реляционные) БД объединяют в себе черты реляционной и объектной моделей. Их возникновение объясняется тем, что реляционные базы данных хорошо работают со встроенными типами данных и гораздо хуже – с пользовательскими, нестандартными. Когда появляется новый тип данных, приходится либо включать его поддержку в СУБД, либо программисту самостоятельно управлять данными в приложении.

Перестройка СУБД с целью включения в нее поддержки нового типа данных – не лучший выход из положения. Вместо этого объектно-реляционная СУБД позволяет загружать код, предназначенный для обработки «нетипичных» данных. Таким образом, база данных сохраняет свою табличную структуру, но способ обработки некоторых полей таблиц определяется извне, т. е. программистом.

7. СУБД и сети

Развитие компьютерных сетей, а тем более рост популярности Интернета, сыграли значительную роль развитии СУБД. Если первые приложение и СУБД находились на одном центральном главном компьютере (*майнфрейме* – от англ. *mainframe* – базовая универсальная вычислительная машина), то теперь они отделены друг от друга: клиентская часть выполняется на компьютере пользователя, а СУБД и собственно сама база данных находятся на сервере. При этом клиент и сервер взаимодействуют друг с другом посредством специального языка запросов.

С развитием Интернета архитектура сетевого управления базами данных получила дальнейшее развитие. Первые сервисы Интернета обеспечивали непосредственны доступ к базам данных. Спустя некоторое время появилась среда WWW, которая представила пользователям Интернета дружественный интерфейс. За формирование этого интерфейса несет ответственность Web-сервер, т. е. на пути между пользователем и базой данных появился еще один посредник, который стал выполнять функции клиентской программы по отношению к БД. Такой подход позволил многочисленным пользователям Интернета иметь единственную программу – Web-браузер – для работы с различными базами данных, будь это интернет-магазин или форум, не прибегая к услугам специальных программ-клиентов. Так как браузер входит в состав любой операционной системы и стандартизован, разработчикам можно не беспокоиться о его установке и обновлениях на клиентских машинах. При этом достаточно изменить код на Web-сервере, чтобы изменения коснулись сразу всех клиентов. Схема организации работы такой трехуровневой архитектуры представлена на рисунке 8.8.

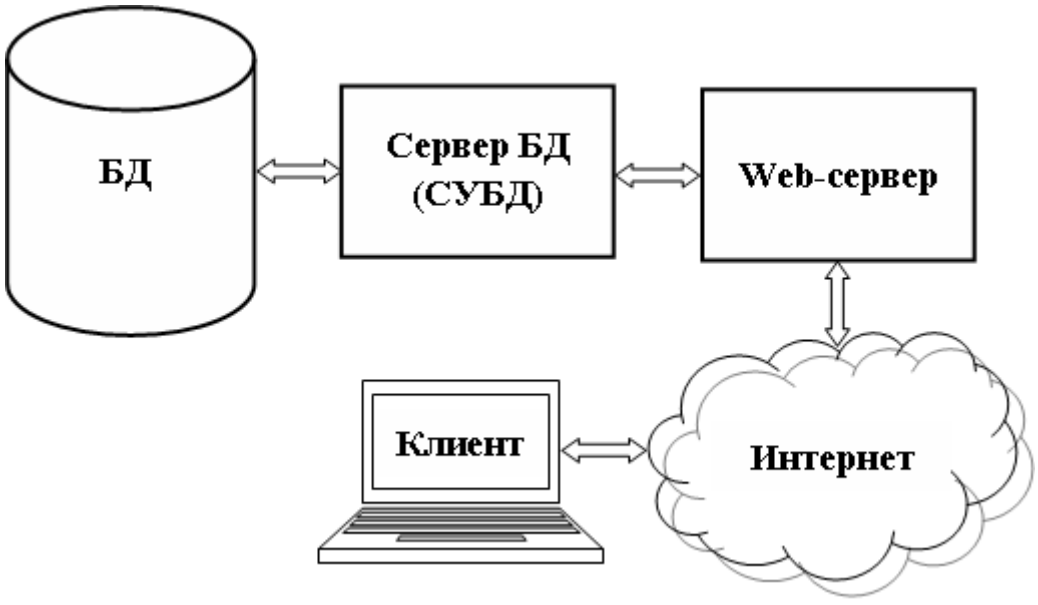


Рисунок 8.8 – Трехуровневая архитектура использования СУБД в Интернете

Такая модель позволяет отделить клиентское программное обеспечение от серверной части, а на серверной стороне отделить Web-сервер от сервера базы данных. Последнее, в частности, является достаточно актуальной проблемой.

Дело в том, что по мере развития Интернета и подключения новых пользователей нагрузка на Web-сайты все время возрастает. И одним из способов решения этой проблемы является увеличение производительности и распределение отдельных серверов.

Схему работы трехуровневой архитектуры можно продемонстрировать на таком примере. Допустим, клиенты некоторого интернет-магазина, находясь за тысячи километров от него, хотят ознакомиться со списком товаров, доступных на данный момент. В этом случае они используют браузер для посещения сайта этого интернет-магазина. Результирующую страницу со списком товаров формирует специальное программное обеспечение, выполняющееся на Web-сервере интернет-магазина. При этом для получения нужной информации это программное обеспечение, в свою очередь, обращается к СУБД, находящейся на сервере баз данных.

Таким образом, в трехуровневой архитектуре *интерфейсом* пользователя является Web-браузер. Браузер взаимодействует с Web-сервером, посылая ему запросы на отображение той или иной Web-страницы. Web-приложение, выполняющееся на Web-сервере – это *прикладной* уровень. Оно формирует запросы к СУБД, которая, в свою очередь, возвращает необходимые данные из БД. СУБД и база данных при этом размещены на сервере баз данных и представляют собой третий, *информационный* уровень трехуровневой архитектуры.

ЛЕКЦИЯ № 9

ОСНОВЫ ОФИСНОГО ПРОГРАММИРОВАНИЯ

План

1. Технологии создания компьютерных программ
2. История создания языка программирования JavaScript
3. Первая программа
4. Линейный вычислительный процесс
5. Разветвляющийся вычислительный процесс
6. Циклический вычислительный процесс

1. Технологии создания компьютерных программ

Компьютерные программы создаются программистами при помощи так называемых *систем программирования*. Каждая из таких систем программирования состоят из 2-х частей:

1. *языка программирования* – набора формальных правил, который предназначен для описания процесса обработки информации на некотором *виртуальном* (*условном*, *гипотетическом,* и т. д.) компьютере, и
2. *интегрированной среды разработки* (*IDE – Integrated Development Environment, среда программирования, транслятор*) – набора программ, предназначенных для *перевода* (*трансляции*) *команд* языка программирования в машинные команды вполне конкретного компьютера.

Таким образом, язык программирования – это средство, с помощью которого программирование ведется на некоторую идеализированную (виртуальную, гипотетическую) вычислительную машину, спроектированную, невзирая на ограничения современных компьютеров но, учитывая традиционные способы и умения человека выражать свои мысли. В результате, в такой ситуации появляется две машины:

- *реальная машина*, создание которой экономически оправдано, но которая не удобна в программировании, и
- *виртуальная машина*, которая вполне согласуется с человеческими нуждами, но «существует только на бумаге».

Роль моста через пропасть, которая разделяет эти два компьютера, выполняют – *транслляторы*. Транслятор (англ. *translator* – переводчик) – это программа для реальной машины, которая дает ей возможность переводить (транслировать) программы, написанные для виртуальной машины, в ее собственные команды. Она позволяет реальной машине выступать в роли виртуальной, идеализированной машины. Применение транслятора, таким образом, освобождает программиста от необходимости рассматривать частные характеристики реального компьютера.

⚠ *Но транслятор не освобождает программиста от обязанности постоянно учитывать тот факт, что в конечном итоге именно реальная машина будет выполнять его программу, и что она имеет определенные ограничения*.

Любая программа на языке программирования состоит из:

1. *инструкций* (*операторов*, *команд*, *предложений* и т. д.), и
2. *комментариев*.

Инструкции служат для описания самого процесса обработки информации, и предназначены для выполнения компьютером. Комментарии же после трансляции никаких машинных команд не порождают. Они предназначены для пояснения наиболее сложных мест программы, и адресованы человеку – т. е. *программы пишутся для компьютеров, а читаются человеком*!

В любом языке программирования имеется два типа инструкций, которые используются для описания:

1. *данных* – информационных объектов, участвующих в процессе обработки – *инструкции-декларации*, и
2. *алгоритмов* – наборов формальных правил, в соответствии с которыми обрабатываются данные – *инструкции-команды*.

Схематически процесс создания компьютерных программ можно представить следующим образом (рисунок. 9.1):

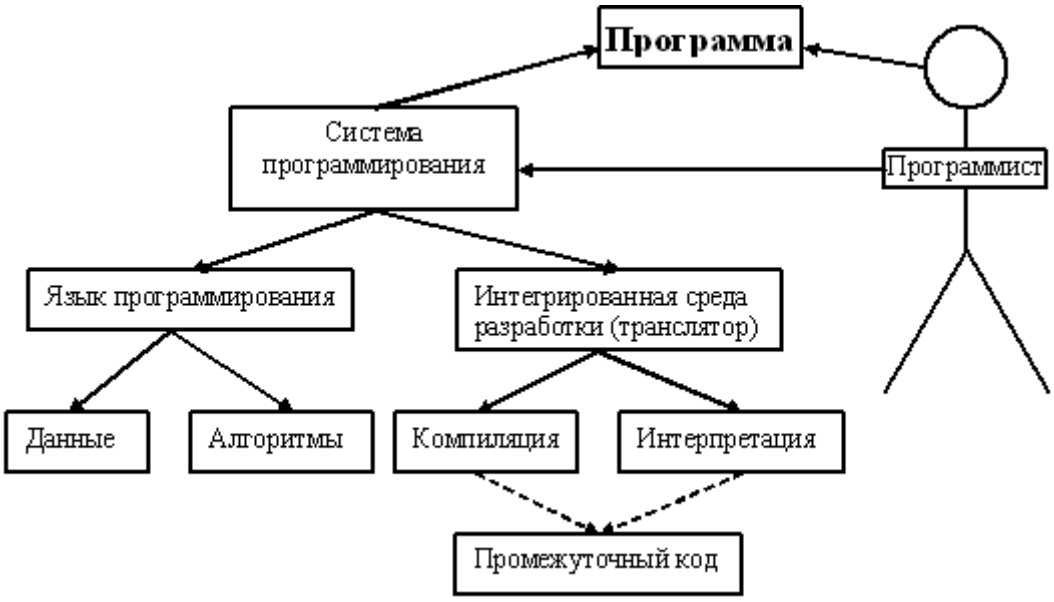


Рисунок 9.1 – Схема создания компьютерной программы

Он принципиально ничем не отличается от процесса приготовления пищи. То есть, для того, чтобы приготовить пищу (например, украинский борщ) необходимы:

1. *повар* – или программист, применительно к компьютерным программам, – т. е. человек, который будет варить борщ или создавать программу,
2. *продукты* – данные, подлежащие обработке,
3. *рецепт* приготовления блюда – алгоритм обработки данных, и
4. *кухонный инвентарь* (кастрюли, ножи, кухонный комбайн и т. д.) – интегрированная среда разработки (транслятор).

Следует также заметить, что процесс построения здания или, вообще, любого другого «творения рук человеческих», также состоит из аналогичных элементов.

Трансляция (перевод) программы с языка программирования в машинные команды совершенно аналогичен переводу с одного естественного языка на другой. При этом существует два вида переводов:

1. синхронный и

2. литературный.

При синхронном переводе переводчик немедленно переводит каждую фразу, как только ее услышит. При литературном переводе он может несколько раз прочитать исходный документ, внимательно его изучить, воспользоваться необходимыми словарями, и лишь затем – подготовить выходной документ на другом языке. Понятно, что качество литературного перевода заметно выше качества синхронного перевода, но им не всегда можно воспользоваться. Так, например, во время международных переговоров или во время демонстрации недублированных фильмов используется синхронный перевод, хотя с литературной точки зрения он не всегда является качественным.

Трансляция (перевод) компьютерных программ с языка программирования в машинные команды также выполняется двумя различными способами. Это:

1. компиляция – аналог литературного перевода и

2. *интерпретация* – аналог синхронного перевода.

Программа-компилятор (англ. *compiler* – составитель, собиратель) работает так же как и литературный переводчик. Сначала она несколько раз внимательно просматривает исходный код программы, потом обращается к необходимым справочникам (которые в программировании называются библиотеками) и лишь затем – выдает готовую программу в машинных командах конкретного компьютера – так называемый *загрузочный* (*выполняемый*, *исполняемый* или *рабочий*) модуль. Созданная таким образом программа в трансляторе больше не нуждается – она может независимо и параллельно с другими программами выполнятся на компьютере.

Работа программы-интерпретатора (англ. *interpreter* – истолкователь, устный переводчик) похожа на работу синхронного переводчика. Она читает исходный текст программы инструкция за инструкцией, переводит их в машинные команды и тут же передает процессору на выполнение. Исполнив, таким образом, одну инструкцию программы, она переходит к другой, и так далее. То есть, программа, написанная программистом, на компьютере выполняется под управлением интерпретатора.

Каждый из этих способов трансляции имеет как свои достоинства, так и вполне определенные недостатки, которые заключаются в следующем:

- Интерпретируемые программы выполняются в несколько раз медленнее, чем откомпилированные. Это расплата за то, что для решения одной задачи компьютер должен одновременно выполнять две программы – исходную программу и «синхронного переводчика», в роли которого выступает интерпретатор. Однажды же откомпилированная программа в дальнейшем не требует присутствия программы-компилятора, и компьютеру больше не нужно «исхитряться», чтобы одновременно и транслировать, и выполнять программу.
- Внесение изменений в интерпретируемые программы выполняется гораздо проще и быстрее, чем в компилируемые, поскольку не требует их повторной перекомпиляции. Интерпретируемые программы после внесения в них изменений можно сразу запускать на выполнение.
- При выполнении программы под управлением интерпретатора имеется возможность контролировать абсолютно все осуществляемые действия, что повышает устойчивость и надежность работы не только конкретной программы, но и всей вычислительной системы в целом. Так, при определенных условиях (а не вообще), программа-интерпретатор может либо запретить, либо, наоборот, разрешить выполнение конкретной программой вполне определенных действий – например, проверить право использования некоторого ресурса.

Каждый конкретный язык программирования, в основном, ориентирован либо на компиляцию, либо на интерпретацию – в зависимости от того, для каких целей он изначально был создан. Так, например, один из первых языков программирования FORTRAN был задуман как язык для разработки больших программ, предназначенных для решения естественнонаучных и математических задач, в которых особенно важна скорость их выполнения. Поэтому данный язык программирования обычно реализуется с помощью компилятора. С другой стороны, BASIC создавался как язык программирования

начинающих программистов, для которых построчное выполнение программы имеет неоценимое преимущество. Иногда для одного языка программирования имеется и компилятор, и интерпретатор. В этом случае для разработки и тестирования программы лучше воспользоваться интерпретатором, а затем отлаженную программу откомпилировать, чтобы повысить скорость ее выполнения.

Существенные достоинства и недостатки каждого из этих методов привели к созданию 3-го, промежуточного, подхода, который называется *компиляцией в промежуточный код*, и который, в некоторой степени, сочетает достоинства двух предыдущих методов – компиляции и интерпретации. При таком подходе:

1. вначале исходная программа компилируется в так называемый *промежуточный код*, который
2. затем выполняется под управлением программы-интерпретатора на конкретном компьютере.

Такое сочетание двух различных подходов к трансляции программ позволяет:

1. уменьшить их размер,
2. повысит скорость выполнения,
3. проконтролировать (при необходимости) выполняемые ими действия, а также
4. получить промежуточный код на одной компьютерной платформе (сочетание компьютера и операционной системы), а выполнить – на другой, что особенно актуально в условиях повсеместного использования Интернета.

Впервые этот подход был реализован в 1995 году компанией Sun Microsystems при разработке языка программирования *Java*. Полученный после компиляции промежуточный код (который называется *байт-кодом*) выполняется под управлением *Виртуальной Машины Java* (англ. *Java Virtual Machine – JVM*). В результате, откомпилированная в

промежуточный код программа на языке Java может выполняться на любой компьютерной платформе, где установлена JVM.

Идея трансляции программы в промежуточный код и ее последующее выполнение под управлением виртуальной машины оказалась довольно удачной. В частности, по такому принципу создаются все приложения для мобильных устройств – смартфонов и планшетных компьютеров и т.д. При этом для выполнения программ на таком устройстве необходимо лишь наличие виртуальной машины Java.

Реализация этого подхода от Microsoft, выпущенная в 2002 году, называется *.NET Framework* (читается «дотНЕТ фреймворк»). Основными языками программирования, которые поддерживает эта платформа, являются Visual Basic, Visual C++ (читается «Визуал Си-плюс-плюс) и Visual C# (читается «Визуал Си-шарп»). Промежуточный код, в который компилируются исходные программы на этих языках, так и называется *IL* (англ. *Intermediate Language* – промежуточный язык). Одной из особенностей Microsoft .NET Framework является совместимость программных частей, написанных на разных языках. Например, программа, написанная на Visual C++, может обратиться к программе, написанной на Visual C#, и – наоборот. Основным ограничением данной платформы является то, что она может работать только под управлением операционных систем от Microsoft. Реализация .NET Framework на базе свободного программного обеспечения для подавляющего большинства популярных ныне операционных систем называется *Mono*.

Идеи компиляции программы в промежуточный код также находит свое воплощение и в IT-решениях для бизнеса. Так, например, наиболее распространенная система *1С:Предприятие* так же создана с использованием этого подхода. В ней программа-интерпретатор называется *платформой*, а откомпилированный промежуточный код – *конфигурацией*. То есть, имеется одна платформа и множество конфигураций для разных законодательных баз и различных предприятий.

Изменения в конфигурации вносят, как правило, программисты, а не конечные пользователи этой системы.

Наиболее бурно развивающийся на сегодняшний день рынок мобильных приложений на базе операционной системы *Google Android* так же использует эту технологию. При этом исходные программы, как правило, пишутся на языке программирования Java. Затем, после компиляции в промежуточный код, они выполняются под управлением программы-интерпретатора, которая в операционной системе Android называется *Виртуальная Машина Dalvik* (англ. *Dalvik Virtual Machine – Dalvik VM*). Виртуальная Машина Dalvik является оптимизированным вариантом JVM. Это, в первую очередь, связано с тем, что мобильные устройства отстают от локальных компьютеров, как по объемам памяти, так и по скорости обработки информации. Поэтому, для достижения требуемой производительности, корпорации Google пришлось во многих отношениях переработать и оптимизировать стандартную версию JVM. Также Dalvik была оптимизирована таким образом, чтобы на мобильном устройстве можно было одновременно запускать несколько приложений. То есть, изнутри работающий Android выглядит как набор виртуальных машин Dalvik, в каждой из которых исполняется своя прикладная задача. Формат промежуточного кода Dalvik также отличается от аналогичного для JVM. Она может выполнять только промежуточный код в формате **DEX** (от. **D**alvik **Ex**ecutable), который получается путем компиляции исходных Java-программ.

Каждому сочетанию:

1. языка программирования,
2. операционной системы и
3. типа компьютера

соответствует своя система программирования. Например, компилятор с языка программирования C++ для операционной системы Windows и компьютера типа IBM PC несовместим с компилятором для того же языка программирования и того же типа компьютера, но для операционной системы

UNIX. Несовместимыми также бывают компиляторы для одного и того же языка программирования, но от разных производителей. Такая несовместимость выражается, главным образом, в различных форматах получаемого после компиляции машинного кода. Поэтому существуют различные сочетания интегрированных сред разработки и языков программирования в зависимости от производителя, типа компьютера и операционной системы, для которых они предназначены. Так, например, интегрированная среда разработки Microsoft Visual Studio предназначена для создания программ на языках Visual Basic, Visual C++ и Visual C# только для компьютеров типа IBM PC, работающих под управлением операционной системы Windows. При этом программу на языке программирования Visual C++ можно откомпилировать как в машинные команды (получить загрузочный модуль), так и в промежуточный код IL. С языков программирования Visual Basic и Visual C# программы можно транслировать только в промежуточный код IL.

Одним из основных конкурентов Visual Studio от Microsoft является *Eclipse*. В отличие от Microsoft Visual Studio, которая может выполняться только под Microsoft Widows и является проприетарной, Eclipse является решением с открытым исходным кодом и может работать практически под управлением любой операционной системы, в том числе, таких как Microsoft Windows, Linux и Mac OS. Поскольку Eclipse написана на языке Java, то для ее выполнения требуется лишь наличие JVM.

Eclipse построена не как единое монолитное приложение, а в виде набора расширяемых подсистем. Сама по себе Eclipse – это только *платформа* для разработки интегрированных приложений. Она предоставляет возможность подключать *дополнения* (*расширения*, *плагины* от англ. *plug-in* – «подключать»), что делает ее весьма мощным инструментом. Именно наличие большого количества бесплатных и коммерческих плагинов обеспечивает возможность разработки не только на Java, но и на других языках, таких как C/C++, Python, PHP, HTML, JavaScript и т. д. Любой разработчик

может создать собственное дополнение для расширения возможностей Eclipse. А в силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.

Одной из основных тенденцией, наметившихся в последнее время в IT-индустрии, являются *облачные вычисления* (англ. *cloud computing*). А поскольку инструментальные средства разработки программ являются неотъемлемой частью этой индустрии, то и у традиционных средств разработки постепенно появляются аналоги в виде Web-сервисов – облачные (англ. cloud) или онлайн-IDE. Например, параллельно с традиционным, настольным, Eclipse существует и онлайн-сервис – *Orion*. Существую также и совершенно самостоятельные решения, наиболее известным среди которых является *Cloud9*. Основным из достоинств облачных IDE являются то, что они совершенно не критичны к параметрам компьютера и операционной системы – достаточно лишь наличие браузера и доступа к Интернету. Они также не требуют установки и настройки дорогого и сложного программного обеспечения (как настольные IDE) – они всегда и в любом месте готовы к работе – за это отвечает Web-сервер. Недостатки же онлайн-IDE являются продолжением их достоинств. И основной из них – это безопасность. А поскольку код доступен из любой точки мира, то не исключается возможность доступа к нему и злоумышленников. Другой немаловажный недостаток всех облачных вычислений заключается в том, что нельзя гарантировать того, что рано или поздно облачный продукт может не прекратит свое существование, и тогда придется решать возникшую задачу безболезненной миграции на альтернативные решения.

Всякая интегрированная среда разработки является совокупностью программных средств, обеспечивающих автоматизацию процесса разработки и отладки программ. Традиционно в ее состав входят следующие компоненты, которые выполняют такие функции:

1. Транслятор (компилятор, интерпретатор или оба вместе) – обеспечивает трансляцию (перевод) программы, написанной на одном из языков

программирования, в машинные команды. Поскольку эта функция является основной, то IDE, и даже всю систему программирования, иногда называют компилятором или транслятором. Никакой путаницы в таких случаях, как правило, не происходит, поскольку из контекста всегда понятно, к чему относится это понятие – к отдельной программе-транслятору, или ко всей системе программирования.

2. Специализированный текстовый редактор – служит для облегчения и, в некоторой степени, автоматизации процесса создания исходных текстов программ путем:
   - выдачи различного рода окон с подсказками,
   - структуризации программы с помощью отступов, и
   - «раскрашивания» различных фрагментов исходного текста программы, в зависимости от их функционального назначения.
3. Компоновщик (он же – редактор связей) – предназначен для компоновки (объединения) отдельных частей большой программы в один загрузочный модуль. Компоновщики используются, как правило, с трансляторами компилирующего типа. Если перевести на «кулинарный» язык, то процесс компоновки – это смешивание всех, отдельно приготовленных, компонентов блюда. Например, заправка украинского борща добрым украинским салом.
4. Отладчик (англ. *d e b u g g e r*) – обеспечивает автоматизацию процесса отладки программ, т. е. поиска и устранения в них ошибок.
5. Оболочка – объединяет в законченную систему все, перечисленные выше, составляющие.

Хорошо спроектированная и реализованная среда программирования не только автоматизирует рутинные операции при создании программ, но и позволяет также быстро получать отличного качества программы, «не выходя» из последней. Она также отслеживает все внесения изменений в исходные тексты программ и, в случае необходимости, перетранслирует только те из них, которые действительно были изменены. При этом для создания (или

перестройки, после внесения изменений) программы необходимо буквально нажать одну-единственную клавишу на клавиатуре, или щелкнуть один раз мышью.

В простейшем случае интегрированная среда разработки для языка программирования JavaScript состоит из двух отдельных компонентов:

1. *текстового редактора* – для создания и редактирования исходных текстов, и
2. *браузера* – для запуска скриптов на выполнение.

В качестве текстового редактора подойдет наиболее простой из них. Например, для операционной системы Linux – это Mousepad, а для Windows – Блокнот. Но скрипты гораздо удобнее создавать и редактировать в специализированном текстовом редакторе. Далее, в качестве текстового редактора будет использоваться интегрированная среда разработки *Geany*, а в качестве браузера – *Mozilla Firefox* потому, что:

1. они имеют версии как для Linux, так и для Windows,
2. у них русский интерфейс и
3. они бесплатные.

2. История создания языка программирования JavaScript

Предпосылкой появления языка программирования *JavaScript* стала проблема, которая возникла в середине 90-х годов прошлого века, и которая была связана с необходимостью сделать Web-странички «живыми». То есть, разработчикам HTML-документов необходим был инструмент, который бы позволял динамически управлять всеми объектами Web-страниц, а также реагирующими на действия пользователя. И такой инструмент был создан Бренданом Айком (Brendan Eich) из Netscape Communications. В разработке языка также принимали участие сооснователь Netscape Communications Марк Андрессен и сооснователь Sun Microsystems Билл Джой.

Впервые новый язык был встроен в 1995 году в браузер Netscape Navigator 2.0 – самый популярный браузер того времени. В дальнейшем к развитию этого

языка подключилась и Microsoft. Так, в 1996 году она выпустила свой вариант языка JavaScript, который получил название *JScript*. А первым браузером, который его поддерживал, был Internet Explorer версии 3.0. Но у фирмы Microsoft возникли определенные трудности использования этого языка связанные с его надежностью. Это привело к тому, что Microsoft в свои браузеры, помимо JavaScript, стал встраивать еще и интерпретатор языка *VBScript* (Visual Basic Script), который является ее авторским решением. Сейчас интерпретатор языка программирования JavaScript встраивается во все основные браузеры, именно поэтому они и могут выполнять скрипты на Web-страницах.

Перед создателем JavaScript, Бренданом Айком, была поставлена задача, реализовать язык, который был бы похож на *Java*, но был поменьше и как писал сам автор эдаким «его младшим братом», и очень важно, имел бы выразительную лексику, чтобы непрограммисты могли быстро освоить новую технологию. Уже по истечении 10-ти дней была написана первая версия языка, которую решили назвать LiveScript. Конечно, создать за такой короткий промежуток времени идеальный язык вряд ли возможно но, несмотря на отдельные недостатки, он решал поставленную перед ним задачу, а именно, позволял динамически изменять Web-страницы без их перезагрузки.

Через некоторое время, язык был переименован в JavaScript. Такое название было выбрано не случайно. Дело в том, что в то время была очень популярна технология Java, и маркетологи решили, что схожее название сделает новый язык более популярным. И это дало свои плоды – люди заинтересовались, попробовали, оценили возможности языка, да так, что после анонса языка в декабре 1995 года, более 20-ти компаний выразили намерение использовать JavaScript в своих будущих продуктах. На синтаксис JavaScript оказали влияние многие технологии и языки программирования, в том числе, такие как C, C++ и Java. Однако, необходимо четко различать, что Java и JavaScript – это совершенно разные языки программирования – со схожим синтаксисом, но различными областями применения.

Компания Netscape Communications сразу после разработки открыто опубликовала все его спецификации – т. е. языком программирования JavaScript не владеет какая-либо компания или организация. Вследствие этого, буквально с самого начала, его реализации в различных браузерах стали сильно отличаться друг от друга – компании, производители браузеров, вводил в него свои, «фирменные улучшения», которых не было у конкурентов. Это, естественно, не обеспечивало совместимости различных «диалектов» языка – одинакового выполнения кода во всех браузерах. Поэтому, по инициативе компании Netscape Communications в 1996 году Европейской ассоциацией производителей компьютеров (European Computer Manufacturers Association – ECMA) была начата его стандартизация, а в июне 1997 года уже была принята первая версия стандарта – ECMA-262. В соответствии с этим стандартом язык стал официально называться $ECMAScript$. Но это несколько неудобное название используется только в случае, если необходимо явно сослаться на стандарт. Чисто технически название «JavaScript» относится только к реализации, выполненной Netscape Communications или ее преемницей – Mozilla Foundation. Однако, на практике все предпочитают использовать именно название JavaScript для обозначения любой его реализации. Справедливости ради, необходимо отметить, что даже на сегодняшний день не все реализации JavaScript полностью соответствуют стандарту ECMA. Но, с каждым днем таких различий становится все меньше и меньше. Сами же названия «Java» и «JavaScript» являются зарегистрированными товарными знаками компании Sun Microsystems (а теперь, после ее поглощения – Oracle Corporation). В апреле 1998 года стандарт ECMA-262 был принят Международной организацией по стандартизации (International Standards Organization – ISO) в качестве международного стандарта под номером ISO-16262.

Хотя изначально и планировалось, что JavaScript будет «младшим братом» Java, но история распорядилась по-иному. По выражению его автора, Брендана Айка, язык делался «за 10 бессонных дней и ночей» (чтобы успеть закончить

работы над ним к выпуску браузера Netscape Navigator 2.0). И потому, что некоторые моменты были продуманы плохо, в нем есть и откровенные ошибки, и вполне объективные недоработки. Есть даже мнение, что JavaScript почти полностью состоит из недостатков. Но, зачастую, недостатки подходов и технологий – это обратная сторона их полезностей. Стоит ли упрекать молоток в том, что он – тяжелый? Да, неудобно, но зато гвозди забиваются лучше. Пока лишь важно знать, что некоторые «особенности» языка не являются чем-то очень критичным, а просто в свое время не были достаточно хорошо продуманы и отлажены. То есть, проблемы здесь нет, а есть свои нюансы. И далее, будем лишь фиксировать внимание на этих недоработках и «граблях»: ничего критичного в них нет, если знаешь – не наступишь. И, в конечном итоге, популярность JavaScript почти никак не зависит от его качества как языка программирования.

Сегодня JavaScript наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности Web-страницам. Однако, область его применения только этим не ограничивается. Он также используется как встраиваемый язык для программного доступа к объектам различных приложений. То есть, JavaScript – это полноценный язык, программы на котором можно запускать не только в браузере, но и на сервере (например, Node.js). Еще он широко используется для написания макросов в таких приложениях от Adobe, как Adobe Photoshop, Adobe Dreamweaver, Adobe Illustrator, а также входящих в состав пакета OpenOffice.org.

3. Первая программа

JavaScript – это объектно-ориентированный язык программирования, который позволяет управлять поведением открытой в браузере Web-страницы. Программы на этом языке называются *скриптами* или *сценариями*. Они выполняется браузером при помощи встроенного в него интерпретатора. Скрипты подключаются напрямую к HTML-документу и, как только Web-страница загружается – тут же выполняются.

Скрипты на Web-страницу можно внедрять несколькими различными способами. Наиболее распространенный из них – при помощи парного тега <script></script>. При этом они могут располагаться как в области head, так и в области body: сценарии всегда обрабатываются и, если необходимо, запускаются на выполнение в том месте HTML-кода Web-страницы, где они присутствуют.

Классическим примером в начале изучения любого языка программирования является программа, выводящая сообщение «Здравствуй, Мир!». Далее создадим сценарий (скрипт) согласно условиям **Примера 1**.

Пример 1

Написать программу (скрипт), которая выводит сообщение «Здравствуй, Мир!».

Для этого в текстовом редакторе подготовим такой код:

```
<html>
  <head>
    <title>JavaScript</title>
    <meta charset=utf-8>

  </head>
  <body>
    <script>
      alert("Здравствуй, Мир!");
    </script>
  </body>
</html>
```

Если код сценария набран без ошибок, то после его сохранения в файле, например, под именем `index.html` и открытия в браузере будет выдано окно сообщения с текстом «`Здравствуй, Мир!`».

Наша первая программа состоит из единственной инструкции – вызова (обращения к) функции `alert()`. Функция `alert()` отображает окно с сообщением и ждет, пока пользователь не нажмет кнопку **ОК**. В качестве параметра она принимает заключенную в кавычки строку символов, которую необходимо отобразить. В программах на JavaScript строковые значения обязательно заключаются в одинарные либо двойные кавычки, чтобы интерпретатор мог отличить их от имен переменных и ключевых слов языка. Такие значения называются *константами строкового* (`string`) *типа*.

Каждая инструкция языка программирования JavaScript, как правило, располагается в отдельной строке кода и заканчивается символом «точка с запятой» (`;`). В этом смысле она напоминает предложение естественного языка – какую-то законченную мысль, в конце которой ставится точка. Отступы и пустые строки в программе необязательны (интерпретатором они не контролируются), но весьма желательны, т. к. они подчеркивают ее структуру и улучшают «читабельность».

В языке программирования JavaScript (в отличие от языка HTML, который не чувствителен к регистру символов) различаются прописные и строчные буквы. Это значит, что имена переменных, функций и другие конструкции языка необходимо записывать так, как приводится в руководствах. ***Все ключевые (служебные, зарезервированные) слова языка программирования JavaScript записываются строчными буквами!*** Например, если вместо `alert()` ввести `Alert()`, то скрипт не отработает и окно сообщения выведено не будут.

Поиск и устранение ошибок подобного рода выполняется двумя способами:

1. «*вручную*» – т. е. более внимательным просмотром исходного кода и

2. при помощи специальных программ – *отладчиков*, например, *Firebug* в браузере Mozilla Firefox.

Первый способ предполагает хорошее знание правил языка программирования. Второй – более быстрый. Но дополнительно необходимо уметь работать с отладчиком. Например, если в Firefox щелкнуть по кнопке  (которая свидетельствует об установленном в нем отладчике Firebug), то внизу основного окна откроется дополнительное окно (рисунок 9.2), на вкладке **Консоль** которого отображается информация об ошибках.

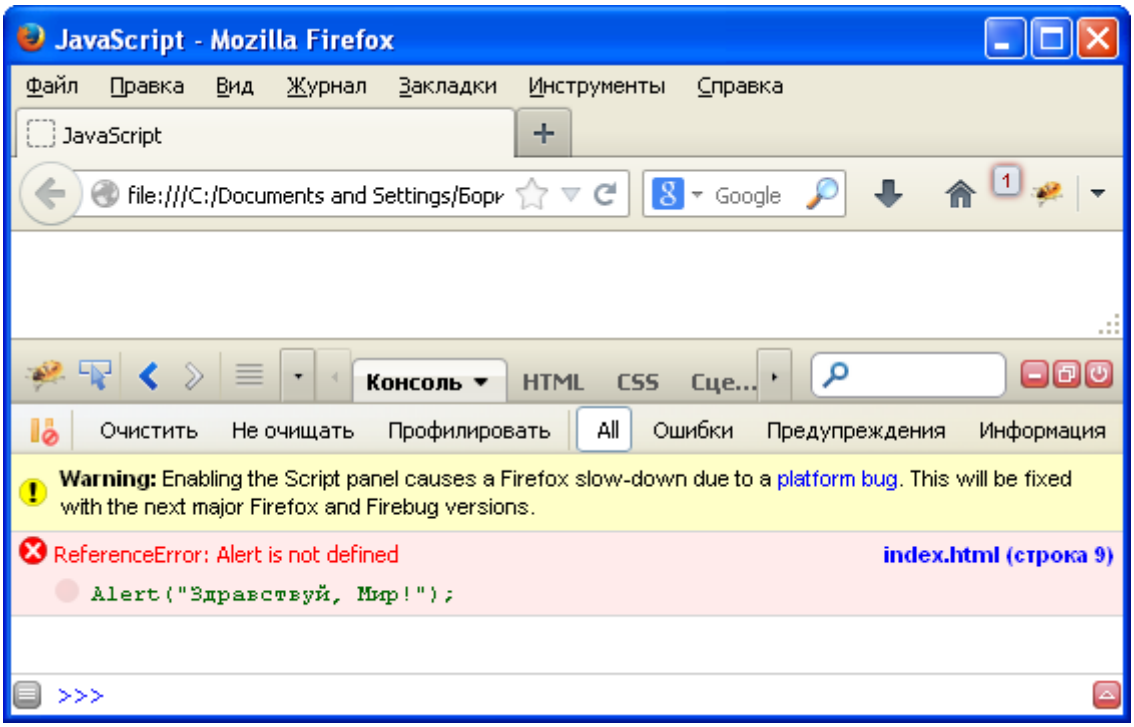


Рисунок 9.2 – Вкладка **Консоль** окна Firebug

Щелчок по строке с ошибкой (которая представляет собой гиперссылку) вызывает переход на вкладку **Сценарии** (рисунок 9.3).

На ней будет отображен весь сценарий, а строка с ошибкой – дополнительно выделена цветом. Редактировать же сценарий на этой вкладке нельзя. Для этого необходимо открыть исходный файл в текстовом редакторе. Например, если отладчик Firebug настроен на использование Geany, то необходимый файл можно открыть при помощи контекстного меню. Причем, в Geany будет выделена именно та строке, по которой был выполнен щелчок правой кнопкой мыши (рисунок 9.4).

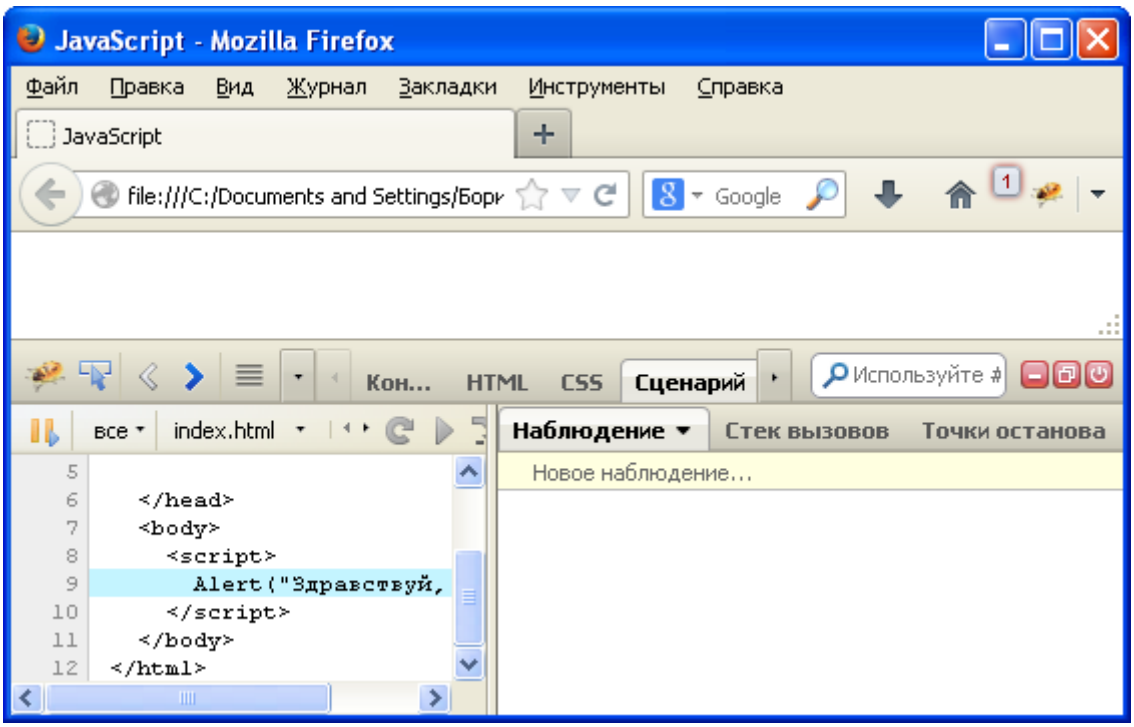


Рисунок 9.3 – Вкладка **Сценарий** окна Firebug

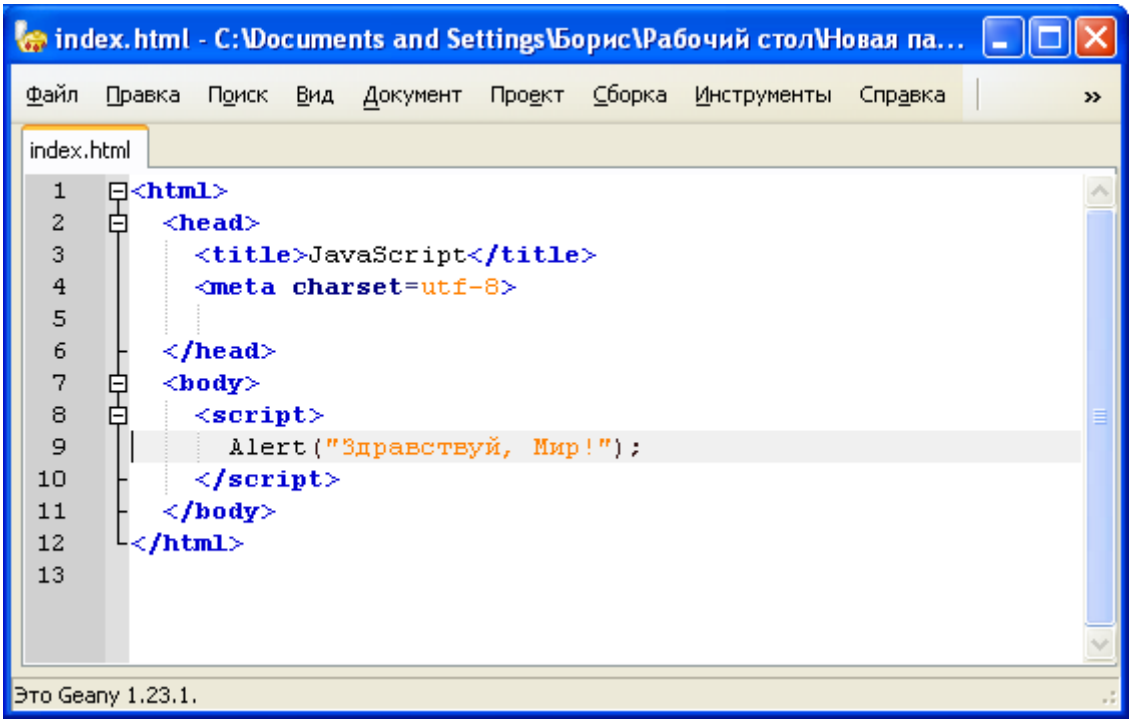


Рисунок 9.4 – Интегрированная Среда Разработки Geany

Тег `<script>` языка HTML имеет два необязательных атрибута:

1. `type=` – в котором стандарт HTML-4 требует обязательного указания того, что скрипт написан именно на языке JavaScript, т. е. `type="text/javascript"`; в новом стандарте HTML-5 этот атрибут не

обязателен – по умолчанию предполагается что сценарий написан именно на языке JavaScript; и

2. `src=` – который содержит ссылку на файл с внешним сценарием.

Изменим нашу первую программу согласно условиям **Примера 2**.

Пример 2

Модифицировать программу из **Примера 1** так, чтобы она при помощи внешнего скрипта приветствовала своего создателя, но уже на английском языке, сообщением «Hello, World!».

Для этого:

1. в том же каталоге, что и основной файл (например, `index.html`) подготовим файл внешнего сценария (например, `script1.js`) такого содержания:

```
alert("Hello, World!");
```

2. в основной файл (например, `index.html`) необходимо внести два изменения:
   a. перенести внутренний скрипт из раздела `body` в раздел `head`;
   б. при помощи атрибута `src=` тега `<script>` подключить внешний сценарий.

   В результате он должен иметь такое содержание:

```
<html>
  <head>
    <title>JavaScript</title>
    <meta charset=utf-8>
    <script src=script1.js>
```

```
      alert("Здравствуй, Мир!");
    </script>
  </head>
  <body>
  </body>
</html>
```

Если коды были подготовлены без ошибок, то после открытия основного файла в браузере будет выдано единственное окно диалога с сообщением «Hello, World!». Содержание внутреннего скрипта (инструкция `alert("Здравствуй, Мир!");`) будет проигнорировано.

Из этого примера видно, что работа с внешними сценариями имеет некоторые особенности, которые заключаются в следующем:

1. по умолчанию внешние файлы сценариев имеет расширение имени файла `.js`;
2. внешний файл сценариев тегов языка HTML содержать не может (например, `<script></script>`) – в нем записываются только команды JavaScript;
3. если подключен внешний файл сценариев, внутреннее содержание тегов `<script></script>` игнорируется.

Последний раз изменим нашу первую программу согласно условиям **Примера 3**.

Пример 3

Модифицировать программу из **Примера 2** так, чтобы выводимое сообщение «Hello, World!» хранилось в переменной.

Для этого необходимо:

1. в основной файл (например, `index.html`) внести два изменения:
   - удалить строку кода

```
alert("Здравствуй, Мир!");
```

которая все равно не работает и

- поменять имя подключаемого скрипта, например, на `script2.js`.

В результате он должен получить такое содержание:

```
<html>
  <head>
    <title>JavaScript</title>
    <meta charset=utf-8>
    <script src=script2.js></script>
  </head>
  <body>
  </body>
</html>
```

В дальнейшем этот файл практически меняться не будет – будут изменяться лишь имена подключаемых скриптов.

2. файл внешнего сценария:
   1) например, `script1.js` сохранить под новым именем, например, `script2.js` и
   2) отредактировать новый файл внешнего сценария (`script2.js`) следующим образом:

```
message = "Hello, World!";
alert(message);
```

Обратите внимание, что в функции `alert()` ***передаваемый параметр – слово*** `message` ***– в кавычки не берется: это имя переменной, а не строковая константа!***

Если теперь в браузере вновь открыть (или обновить) основной файл (`index.html`), то внешне ничего не изменится – будет выдано тоже, что и ранее, окно диалога с сообщением «`Hello, World!`». В этом примере было продемонстрировано, что:

1. в некоторых местах программы, в частности, при обращении к функции `alert()` вместо строковой константы (`"Hello, World!"`) можно использовать переменную (`message`) и
2. одну и ту же программу можно написать множеством различных способов. А какой же из них лучший? – станет понятным с приобретением личного опыта…

4. Линейный вычислительный процесс

Действия, которые программа (скрипт) выполняет над исходными данными, образуют так называемый *вычислительный процесс* (*алгоритм*). Он описывается последовательностью из инструкций-команд. Выполняя такие инструкции-команды одну за другой, виртуальная JavaScript-машина производит необходимые действия по обработке информации.

Любая программа, на любом языке программирования, любой степени сложности, состоит из различных сочетаний следующих 3-х базовых алгоритмов:

1. *линейного*,
2. *разветвляющегося* и
3. *циклического*.

Линейный алгоритм является простейшим вычислительным процессом. В нем все действия по обработке информации выполняются от начала и до конца программы строго последовательно. Такой порядок выполнения действий называется естественным. То есть, для реализации линейного алгоритма ни

каких специальных дополнительных управляющих конструкций в программу вводить не нужно.

Пример 4

Написать программу (скрипт), которая вычисляет сумму двух чисел, например, 2 и 3.

Для этого необходимо подготовить и подключить к основному файлу внешний скрипт (например, `script3.js`) такого содержания:

```
a = 2;
b = 3;
c = a + b;
alert(c)
```

И если он был создан без ошибок, то после открытия (или обновления) основного файла в браузере будет выдано окно диалога с результатом – 5.

Вновь созданный скрипт представляет собой типичный линейный алгоритм. В нем все действия по обработке информации выполняются от начала и до конца программы строго последовательно. То есть, в начале, переменной `a` присваивается значение `2`; затем, переменной `b` – значение `3`; далее, значения переменных `a` и `b` складываются, а результат присваивается переменной `c`; и, наконец, при помощи функции `alert()` выводится результат вычисления – значение переменной `c`. На этом выполнение программы завершается.

Элементы внутренних данных, которые используются в программах, бывают двух видов. Это:

- *переменные* и
- *константы*.

Всякая переменная характеризуется тремя параметрами:

- *именем*,
- *типом* и
- *значением*.

Имя переменной выбирается программистом произвольно, но, по-возможности, должно максимально точно соответствовать ее «смыслу». Оно может содержать любое количество латинских символов и цифр – ***без символа пробела и начинаться с «буквы»***; ***имя переменной, также, не должно совпадать с ключевыми словами языка (например, таким как*** `var`***)***. Если имя переменной состоит из нескольких слов, то применяется так называемая «верблюжьяНотация» – когда каждое слово (кроме первого) начинается с большой буквы. Имя переменной, как правило, – существительное.

Тип переменной определяет множество значений, которые эта переменная может принимать, и операции, которые над ней могут выполняться. В языке программирования JavaScript определены следующие три основные (элементарных) типа данных:

- *логический* (`boolean`), который содержат всего два значения: `true` и `false`. Над ними можно выполнять такие операции, как:

  `&&` *логическое И* – возвращает `true`, если оба операнда имеют значение `true`.

  `||` *логическое ИЛИ* – возвращает `true`, если хотя бы один операнд имеют значение `true`.

  `!` *логическое НЕ* – унарная операция – возвращает `true`, если ее операнд имеют значение `false`.
- *строковой* (`string`) – самые распространенные на Web-страницах значения. К нему по умолчанию преобразуются другие типы данных в разнотипных операциях (если, конечно, нет явных указаний о преобразовании). Над строковыми данными можно выполнять единственную операцию – «+» – конкатенацию («склеивание», объединение) строк. В программе на JavaScript строковые значения заключаются в

одинарные либо двойные кавычки (чтобы интерпретатор мог отличить их от имен переменных и ключевых слов языка). При этом ***не допускается вложенность одинаковых кавычек!***

- *числовой* (`number`), который включают как целые, так и дробные значения. Над ними можно выполнять следующие операции:

+ сложение,

- вычитание,

***** умножение,

/ деление,

% деление по модулю (остаток от деления нацело).

Операция деление по модулю `%` выполняется только над целочисленными значениями. Например, `7 % 2 = 1`.

Значения переменных по ходу выполнения программы могут меняться – потому они так и называются – переменные. Тип переменной также может меняться, а вот имя – нет. Поэтому язык программирования JavaScript является *слабо типизированным* языком или языком *с динамической типизацией*. Это значит, что одна и та же переменная по ходу выполнения программы, в различные моменты времени, может содержать значения различных типов. Или, более формально, тип переменной становится известным к моменту ее использования. То есть, интерпретатор JavaScript всегда знает, какого типа значение содержит данная переменная, а значит – какие операции к ней применимы. Есть *строго типизированные* языки программирования или языки *со статической типизацией* (например, C), в которых тип переменной указывается в момент ее создания – в инструкции-декларации. Есть языки программирования, которые одновременно поддерживают и статическую и динамическую типизацию, например, Java. ***Язык программирования JavaScript поддерживает только динамическую типизацию!***

Переменную в программировании можно сравнить с одной из коробок, которые хранятся у Вас в кладовке. В одной из таких коробок хранятся туфли, в

другой – кроссовки, в третьей – еще что-то. По аналогии с переменными, содержание коробки – это значение переменной. Для того, чтобы знать, что хранится в каждой коробке, они должны быть подписаны. А чтобы не запутаться – надписи на коробках должны быть уникальными. Поэтому, надпись на коробке – это имя переменной. Если туфли – это обувь, а кроссовки – это спортивная обувь, то по аналогии с переменными – это тип данных, который указывает, «какие операции над ними можно выполнять». То есть, туфли мы одеваем, когда идем на работу или в ресторан, а кроссовки – кода в спортзал. По аналогии, числовые данные можно складывать, умножать, делить и т. д. Над строковыми данными таких операций выполнять нельзя – их можно только конкатенировать («склеивать», объединять).

Объекты внутренних данных программ, которые по ходу выполнения программы своих значений не меняют, (т. е., могут использоваться только справа в операции присваивания «=») называются *константами*. Они, так же как и переменные, характеризуются именем, типом и значением. Но, в отличие от последних, своих параметров в ходе выполнения программы не изменяют. Поэтому они и так называются константами (постоянными, неизменяемыми) Параметр имя для констант является необязательным. Вследствие этого, они бывают:

- *неименованные* и
- *именованные*.

Далее приведены примеры неименованных констант:

- `2` и `3.5` – целая и дробная числовые константы (для отделения целой и дробной части числа в программировании используется точка, а не запятая, как принято в математике);
- `"Это пример строковой константы"` и `'Это тоже'` – они заключаются в одинарные или двойные кавычки;
- `true` и `false` – логические константы имеют только два значения.

В языке программирования JavaScript нет специальных средств для работы с именованными константами. Однако, по общепринятой практике, для того,

чтобы визуально отличить именованную константу от переменной, их имена записываются только ПРОПИСНЫМИ_СИМВОЛАМИ, а отдельные слова в имени разделяются символом подчеркивания «_». В программировании именованные константы, в основном, используются в двух случаях:

- если одно и тоже значение в программе используется несколько раз, то имеет смысл создать именованную константу – чтобы в разных местах по-ошибке не присвоить различные значения, и
- для того, чтобы по программе небыли разбросаны различные «магические числа», значения которых не всегда очевидны из контекста. Например, если в начале программы создать именованную константу `WIDTH_WINDOW = 1024;`, то где-то в другом месте программы выражение `a = WIDTH_WINDOW;`, будет более наглядным и понятным, нежели просто `a = 1024;`, – переменной `a` присваивается именно ширина окна, а не какое-то непонятное число `1024`.

Если для вывода значений переменных в JavaScript используется функция `alert()`, то для ввода их значений – функция `prompt()`. Она имеет один обязательный параметр – текстовую строку-приглашение, которая отображается в окне ввода. Возвращает она введенные данные ***в виде значения строкового типа*** – если пользователь нажал кнопку **OK**, или `null` – если он нажал кнопку **Отмена**. Для демонстрации возможностей ввода-вывода языка JavaScript рассмотрим **Пример 5**.

Пример 5

Написать программу, которая эхом отображает введенную в нее информацию.

Для этого проще всего, взяв за основу скрипт из **Примера 3** (файл `script2.js`), сохранить его под новым именем (например, `script4.js`) и отредактировать следующим образом:

```
message = prompt("Введите данные");
alert(message);
```

Если при его подготовке не были допущены ошибки, то при выполнении будет выдано окно диалога со строкой-приглашением "Введите данные". Введенные пользователем в этом окне диалога данные, после нажатия кнопки **OK**, при помощи функции prompt() будут присвоены переменной message. Далее, при помощи функции alert(), значение переменной message будет выведено. То есть, наша программа «работает как попугай» – что в нее ввели, то она и выведет – и является хорошим примером, который демонстрирует возможности ввода-вывода языка программирования JavaScript, но не все ...

Поэтому, далее, с целью более досконального освоения особенностей ввода-вывода языка программирования JavaScript, рассмотрим следующий **Примет 6**.

Пример 6

Написать программу, которая вводит два числа, а выводит – их сумму. Ввод исходных данных и вывод результата выполнить с помощью окон диалога.

В программировании редко какая программа пишется «с нуля». Гораздо чаше «за основу» берется некоторый прототип и уже он «доводится до ума». Поэтому, далее взяв за основу script3.js и сохранив его под новым именем, например, script5.js отредактируем последний следующим образом:

```
a = prompt("Введите a");
b = prompt("Введите b");
c = a + b;
alert(c);
```

При этом все изменения нашего скрипта заключались лишь в том, что числовые константы `2` и `3` были заменены на обращение к функции `prompt()`, с помощью которой можно вводить вообще любые значения, а не только `2` и `3`.

Если теперь, обновив окно браузера, запустить скрипт на выполнение, и в раскрывающихся окнах диалога ввести значения `2` и `3`, то результат получим неожиданный – `23`. Поиск и устранение этой ошибки будем вести при помощи отладчика Firebug. Для этого запустим его, щелкнув в браузере Firefox по кнопке  , и перейдем на вкладку **Консоль**, где отображается информация об ошибках. Однако, никаких ошибок там не зафиксировано! Даже, если повторно запустить скрипт на выполнение и ввести другие данные, результат все равно будет неверным. То есть, скрипт работает не правильно. Поэтому необходимо, хотя бы чуть-чуть, освоить теорию поиска и устранения ошибок.

Все ошибки, которые встречаются в программах, делятся на две категории:

- *синтаксически*, которые нарушают синтаксис языка и не позволяют программе вообще далее выполняться (например, пропущена парная кавычка, используется необъявленная переменная в «строгом» режиме и т. п.); сообщения об таких ошибках, как правило, выводятся на вкладку **Консоль** Firebug (например, ошибка из рассмотренного ранее **Примера 1**), и
- *семантические* (времени выполнения, «прокол в сюжете») – они не блокируют выполнение программы, но она работает не правильно (например, некорректное применение типов данных); для поиска и устранения таких ошибок наиболее часто применяется трассировка – выполнение всего или части скрипта в пошаговом режиме с наблюдением динамики изменения переменных.

То есть, в нашем конкретном случае, мы имеем дело с семантической ошибкой (типичный «прокол в сюжете»). Практическая локализация и устранение подобных ошибок выполняется в пошаговом режиме работы программы с наблюдением за динамикой изменения значений переменных. Для запуска скрипа не в автоматическом (как обычно), а в пошаговом режиме, в

Firebug необходимо перейти на вкладку **Сценарий** и установить там, на нужных инструкциях одну, или несколько, точек останова (англ. *breakpoints*). При достижении потока управления скриптом помеченной инструкции он перед ней остановится. В этой точке программы можно посмотреть значения переменных и продолжить выполнение скрипта далее как в пошаговом, так и в автоматическом режиме. Установка (а также снятие) точек останова выполняется щелчком по номеру нужной инструкции слева на вкладке **Сценарий** окна Firebug.

Установим точку останова нашего скрипта на первой строке кода – инструкции `a = prompt("Введите a");` – и повторно запустим его на выполнение, обновив окно браузера. При этом выполнение сценария будет остановлено на помеченной инструкции (рисунок 9.5).

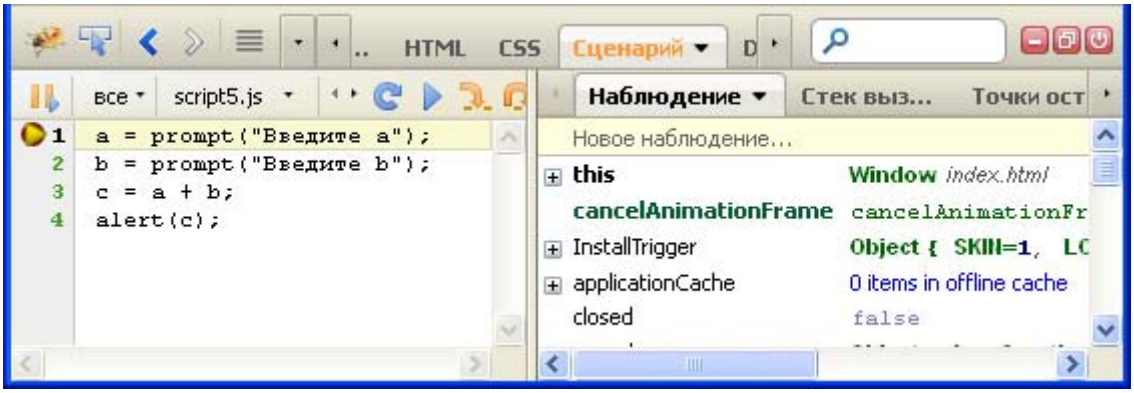


Рисунок 9.5 – Точка останова в отладчике Firebug

Если щелкнуть по кнопке **Шаг с обходом (F10)** или **Шаг с заходом (F11)** (на данном этапе обучения разницы никакой нет), или нажать соответствующую клавишу, то выполнится текущая, помеченная, инструкция – `a = prompt("Введите a");` – будет выведено окно диалога для ввода значения переменной `a`. После ввода значения переменной `a`, например, `2` и щелчка по кнопке **OK**, окно диалога закроется, а процесс выполнения остановится на следующей инструкции – `b = prompt("Введите b");` – ввод значения переменной `b`. ***А дальше – самое интересное!!!***

Наведем указатель мыши на переменную `a` в строке `1`, и внимательно рассмотрим окно Firebug (рисунок 9.6).

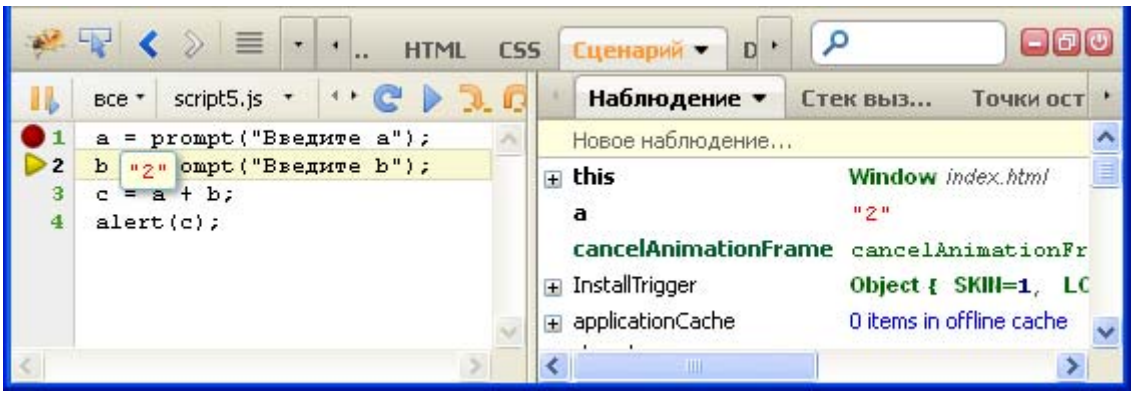


Рисунок 9.6 – Просмотр переменных в отладчике Firebug

Оно состоит из двух дочерних окон: в левом из них отображается вкладка **Сценарий** – исходный код текущего скрипта, а в правом – **Наблюдение** – значения переменных. При этом переменная `a` (и на вкладке **Наблюдение** и в «подсветке» в левом окне) имеет значение `"2"`. ***Обратите внимание!!! – В кавычках???*** То есть, она содержит значение строкового типа (`string`), а должна – числового (`number`). Ура! Мы нашли ошибку! Осталось ее устранить…

Но, перед исправлением, необходимо до конца выяснить причину возникновения ошибки. А она заключается в том, что сценарий отработал так, потому что:

1. функция `prompt()` возвращает значение строкового типа,
2. оператор «+» (строка кода `3`) имеет смысл как для числовых значений – операция сложение, так и для строковых – операция конкатенация.

То есть, исправление ошибки состоит в том, что к моменту *использования* (строка кода `3`) переменные `a` и `b` должны иметь числовой тип. При этом есть два варианта решения:

- «правильный» и
- «быстрый».

«Правильный» вариант состоит в применении одной из функций языка программирования, например, `parseFloat()`, для преобразования строкового значения в число, а второй – в использовании унарного оператора «+».

Для преобразования переменной a из строкового типа в числовой при помощи функции parseFloat() необходимо перед строкой вычисления результата (c = a + b;) вставить такую инструкцию:

```
a = parseFloat(a);
```

В ней, в начале, функции parseFloat() в качестве параметра передается переменная a. Затем, функция parseFloat() переданный ей параметр строкового типа преобразует в числовой. И, наконец, результат, который возвращает функция parseFloat() присваивается переменной a, затирая ее старое содержание. В итоге, в той же переменной из ее строкового значения получается числовое. Сам же отредактированный скрипт (файл script5.js) должен иметь такое содержание:

```
a = prompt("Введите a");
b = prompt("Введите b");

a = parseFloat(a);
c = a + b;

alert(c);
```

Дополнительные пустые строки в нем ни как не влияют на его выполнение, а служат лишь для лучшего зрительного восприятия. Они отделяют друг от друга его основные функциональные блоки. А блоки эти встречаются в любой компьютерной программе. Это:

1. ввод исходных данных,
2. их обработка и
3. вывод результата.

Если выполнить теперь скрипт в пошаговом режиме, и понаблюдать за изменение переменных, то можно увидеть (рисунок 9.7), что переменная `a` к моменту ее использования в инструкции `c = a + b;` имеет значение числового типа – `2`.

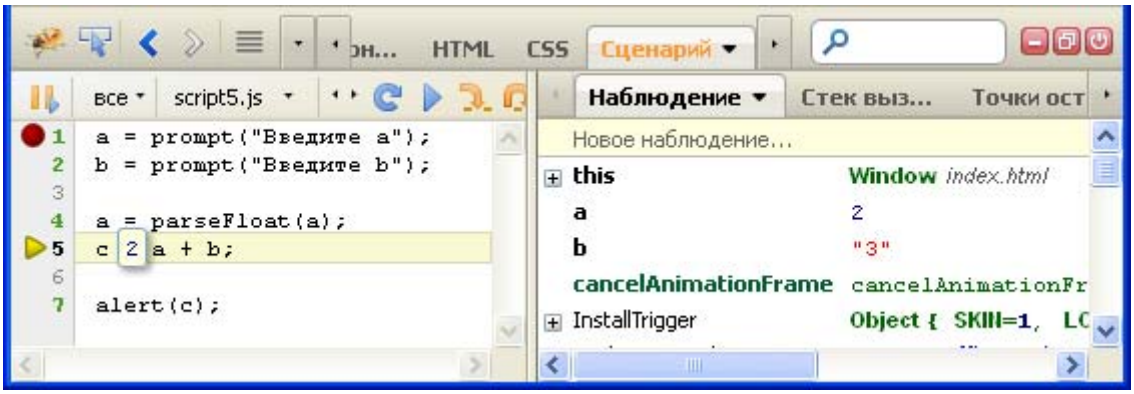


Рисунок 9.7 – Значения переменных перед сложением

Если же скрипт выполнить до конца (нажав, например, кнопку  **Продолжить (F8)** на вкладке **Сценарий** Firebug чтобы продолжить его выполнение не в пошаговом режиме, а в автоматическом), то результат будет прежним. То есть, мы опять получим `23` вместо ожидаемых `5`. Так происходит потому, что в инструкции вычисления результата `c = a + b;` используются два разнотипных операнда: `2` – числового типа и `"3"` – строкового. Интерпретатор JavaScript в такой ситуации использует правило, согласно которому, ***если при выполнении оператора «+» один из операндов строкового типа, то и второй тоже преобразуется к строковому.*** То есть, была выполнена операция конкатенации (`2 +"3" = "23"`) вместо необходимого сложения (`2 + 3 = 5`).

Исправим данную ошибку (преобразуем вторую переменную, `b` из строкового типа в числовой) «по-быстрому» – при помощи унарного оператора «+». Для этого в строке `2`, перед именем функции `prompt()`, поставим унарный «+»:

```
b = +prompt("Введите b");
```

Теперь наш скрипт будет работать как надо, потому что к моменту использования в инструкции `c = a + b;` обе переменные имеют числовой тип, хотя после ввода с помощью функции `prompt()` они мели строковой.

Появление ошибок подобного рода в языке программирования JavaScript является следствием его слабой (динамической) типизации. Аналогичные ошибки в языках программирования со статической (строгой) типизацией (например, C) обнаруживаются еще на этапе трансляции. То есть, они там относятся к категории синтаксических ошибок и обнаруживаются гораздо легче и проще, чем семантические, – в случае синтаксической ошибки, как правило, выдается соответствующее диагностическое сообщение. Однако, в JavaScript такое положение вещей является своеобразной «платой» за его исключительную простоту.

Тот же результат можно получить и другим способом – поставив унарный «+» перед переменной `b` непосредственно в самом выражении:

```
c = a + +b;
```

А вот ставить унарный «+» перед переменной `b` в инструкции ввода:

```
+b = prompt("Введите b");
```

нельзя. Скрипт работать не будет – ***слева в инструкции (операторе) присваивания «=» может стоять только имя переменной***. Более подробно текст сообщения, связанного с этой ошибкой, можно посмотреть на вкладке **Консоль** Firebug, а исправить – в Geany.

Присваивание – одно из самых фундаментальных действий в программировании. Инструкция присваивания делится на две части символом «=», который является знаком операции присваивания. Та часть, которая находится справа от него, называется *выражением*, а часть, расположенная слева, определяет *имя переменной*, которой должно быть присвоено *значение*,

полученное после вычисления выражения. ***Операция присваивания всегда выполняется справа налево***, так что выражение `20 = x` вызовет ошибку, поскольку предпринимается попытка присвоить `20` новое значение. Такое действие невозможно, поскольку `20` не является переменной, а есть целая числовая неименованная константа, значение которой не может быть изменено.

Еще одним источником потенциального появления семантических, трудно обнаруживаемых ошибок, является способ создание (объявления, декларации) переменных. В языке программирования JavaScript переменные можно создавать двумя способами:

- *явно* – при помощи инструкции-декларации `var` и
- *неявно* – фактом своего первого появления в программе.

Все переменные, во всех рассмотренных ранее Примерах, создавались неявно. Такой способ хоть и кажется более простым, уступает первому в смысле потенциального появления трудно обнаруживаемых ошибок (например, непреднамеренное использование переменной, которая уже задействована где-то ранее в другом контексте). Поэтому в последних версиях языка введена специальная *инструкция-директива* `"use strict";`, которая, помимо других «строгостей», «заставляет» интерпретатор выдавать сообщение об ошибке в случае использования необъявленной переменной.

Явно переменные можно создавать в любом месте программы, даже после их использования (поскольку, интерпретатор «просматривает» код как минимум два раза), но лучше – в начале. После создания (объявления) переменной при помощи инструкции `var` она принимает специальное значение (тип) `undefined`. В этой же инструкции `var` можно также присвоить переменной некоторое *начальное значение* (*инициализировать*) – тогда она сразу будет иметь не только имя, но и свой тип и значение. При этом распределение переменных по декларациям является произвольным, т. е. можно сосредоточить все переменные в одной декларации, но можно и, наоборот – для каждой переменной своя декларация. Руководствоваться всегда

нужно здравым смыслом, и читабельностью программы, если нет на то других соображений.

Далее, на **Примере 7**, более подробно рассмотрим особенности создания переменных в языке программирования JavaScript.

Пример 7

Переписать программу из **Примера 6,** обеспечив в ней явное создание переменных.

Для этого, на основании файла `script5.js`, создадим новый сценарий, например, `script6.js` и отредактируем его следующим образом:

```
"use strict";

var a, b, c;

a = prompt("Введите a");
b = +prompt("Введите b");

a = parseFloat(a);
c = a + b;

alert(c);
```

Если теперь запустить его на выполнение, то внешне ни чего не изменится – он, как и прежде, будет складывать два введенных числа. Однако, такая «конструкция» скрипта сделала его более надежным. Например, если мы, «забудем» объявить переменную `c`, то скрипт работать не будет. А сообщение об ошибке «`ReferenceError: assignment to undeclared variable c`», как всегда, можно увидеть на вкладке **Консоль** Firebug. Оно

говорит о том, что это ошибка связана со ссылкой, а если точнее – используется необъявленная переменная `c` (в инструкции вычисления результата `c = a + b;`). То есть, применение в самом начале скрипта инструкции-директивы `"use strict;"`, «переводит» ошибки подобного рода в разряд синтаксических, которые относительно легко обнаруживаются и устраняются.

Код программы на языке JavaScript, помимо инструкций, может содержать и *комментарии*. При интерпретации программы они, пропускаются и совершенно не влияют на ее выполнение. Комментарии адресованы тем, кто будут читать данную программу, и поясняют основные моменты ее реализации. Комментарии в языке программирования JavaScript бывают двух видов:

- *однострочные*, которые начинаются двумя наклонными черточками «//» и продолжаются до конца текущей строки;
- *многострочные и внутристрочные*, которые начинаются с символов «/*», а заканчивается – «*/», между которыми можно вставлять любой текст. Они могут быть записаны в любом месте программы, где можно поставить пробел. Единственное ограничение в многострочных комментариях – они не должны быть вложенными, как, например:

```
/* Внешний /* Внутренний комментарий */ комментарий */
```

Поскольку код JavaScript может быть вставлен в Web-страницу, ***необходимо четко отличать комментарии в языке JavaScript от комментариев языка HTML*** (`<!-- Комментарий HTML -->`)!

Использование комментариев в сценариях на языке программирования JavaScript рассмотрим на **Примере 8**.

Пример 8

Отредактировать скрипт из **Примера 7,** введя в него комментарии.

Для этого, на основании сценария `script6.js`, создадим `script7.js` такого содержания:

```
"use strict"; // Строго! Все под контролем!

//Объявление переменных
var a, b, c;

/* Ввод исходных данных
*  и преобразование в числовой тип */
a = prompt("Введите a");
b = +prompt("Введите b");

// Обработка данных
a = parseFloat(a); // Преобразование в число
c = a + b;         // Вычисление результата

// Вывод результата
alert(c);
```

Функционально он ни чем не отличается от своего непрокомментированного предшественника. Однако, такой текст обладает, по крайней мере, двумя преимуществами:

- если через две недели сам автор еще может вспомнить, что и как делает код, то через полгода – с трудом; и тут ему помогут комментарии;
- не автор текста, например, программист, который будет его модифицировать, быстрее и легче поймет, как он работает, т. е. такой код легче в сопровождении.

Завершая текущий раздел, рассмотрим еще одну, ставшую уже классической, программу обмена значениями двух переменных – **Примера 9**.

Пример 9

Написать программу обмена значениями двух переменных. Ввод исходных данных и вывод результата выполнить с помощью окон диалога.

Условия этого примера очень похожи на пример из реальной жизни: кода один стакан содержит, например, молоко, а в другой – подсолнечное масло, и нужно содержимое этих стаканов обменять. То есть, необходимо в стакан, где было молоко, перелить подсолнечное масло, а в стакан, где было подсолнечное масло – молоко. Очевидно, что в этой ситуации не обойтись без третьего, вспомогательного, стакана. Сама же процедура также состоит из трех шагов:

1. в начале необходимо, например, в третий, вспомогательный, стакан перелить молоко,
2. затем – в освободившийся стакан вылить подсолнечное масло, и
3. наконец – в освободившийся стакан из подсолнечного масла вылить содержимое третьего, вспомогательного, стакана – молоко.

Ситуация с переменными в программировании – аналогичная. Пусть, например, мы имеем две исходные переменные: `a` и `b`. Тогда, алгоритм обмена значениями двух переменных с использованием третьей, вспомогательной, переменной, например, `c` аналогичен предыдущему:

1. в начале содержимое одной из переменных, например `a`, присваивается третьей, вспомогательной, переменной `c`: `c = a;` (теперь содержимое переменной `a` можно «затирать» – оно уже сохранено в переменной `c`);
2. далее – переменной `a` присваивается значение переменной `b`: `a = b;` (содержимое переменной `b` можно «затирать» – оно переписано в переменную `a`) и
3. наконец – переменной `b` присваивается значение третьей, вспомогательной, переменной `c`: `b = c;` содержимое третьей, вспомогательной, переменной

`c` нас больше не интересует – задача выполнена – значения переменных поменялись местами.

Реализация этого алгоритма в виде сценария на языке программирования JavaScript выглядит следующим образом – файл `script8.js`:

```
"use strict"; // Строго! Все под контролем!

//Объявление переменных
var a, b, c;

// Ввод исходных данных
a = prompt("Введите значение первой переменной");
b = prompt("Введите значение второй переменной");

// Обмен значениями переменных
c = a; // Первый шаг алгоритма
a = b; // Второй шаг алгоритма
b = c; // Третий шаг алгоритма

// Вывод результата
alert(a);
alert(b);
```

Если запустить этот сценарий на выполнение, то он в начале выведет значение переменной, которая была введена второй, а затем – той, которая первой. То есть, он работает точно в соответствии с поставленной задачей, но – совсем не наглядно. Далее, изменим его в соответствии с условиями **Примера 10**.

Пример 10

Изменить скрипт **Примера 9** так, чтобы он в начале выводил исходное состояние переменных, а после перестановки - результирующее.

Для этого сохраним его под именем `script9.js` и отредактируем таким образом:

```
"use strict"; // Строго! Все под контролем!

//Объявление переменных
var a, b, c;

// Ввод исходных данных
a = prompt("Введите значение первой переменной");
b = prompt("Введите значение второй переменной");
// Вывод исходных данных
alert("Исходное состояние переменных\n" +
      "a = " + a + "      b = " + b);

// Обмен значениями переменных
c = a; // Первый шаг алгоритма
a = b; // Второй шаг алгоритма
b = c; // Третий шаг алгоритма

// Вывод результата
alert("Результирующее состояние переменных\n" +
      "a = " + a + "      b = " + b);
```

Основное отличие этого кода от предыдущего – в вызове функции `alert()` для отображения исходного и результирующего (после перестановки) состояния переменных. На этом моменте остановимся более подробно.

Как нам уже известно, из предыдущего материала, единственный аргумент функции `alert()` – это значение строкового типа. Если же оно не строкового типа – то будет преобразовано к нему. Единственной операцией, которую можно выполнять над значениями строкового типа, является конкатенация (объединение, «склеивание») строк. Специальная комбинация символов «`\n`» – обратный слеш и латинская строчная буква `n` – вызывает «разлом» строки в месте своего появления, т. е. вся дальнейшая информация будет выводиться с новой строки. Тогда, в результате работы, например, последней функции `alert()` будет выведена строка текста «`Результирующее состояние переменных`», а дальше, с новой строки – текст «`a = `» и значение самой переменной `a`, приведенное к строковому типу, и, наконец – текст «` b = `» (с пробелами перед `b`, чтобы зрительно отделить его от предыдущего текста) и значение переменной `b`.

Наиболее наглядно работу программы можно проследить, запустив ее на выполнение в режиме *трассировки* (*пошаговом режиме*). При этом процесс выполнения программы можно представить кратко и наглядно в виде так называемой *трассировочной таблицы*. Она отображает действия, производимые инструкциями программы по ходу ее выполнения, и изменения значений переменных, вызванные этими действиями.

Например, если установить точку останова на строке `4` – первой инструкции скрипта – инструкции объявления переменных – `var a, x, c;` – и выполнить программу в пошаговом режиме с такими значениями исходных данных: `a = 2`, `b = 3`, то трассировочная таблица для программы обмена значениями переменных будет иметь следующий вид:

Строка	Переменные			Действия
	a	b	c	
4	?	?	?	Объявление переменных
7	?	?	?	Ввод значения переменной `a`
8	2	?	?	Ввод значения переменной `b`
11	2	3	?	Вывод исходного состояния переменных
14	2	3	?	Присвоить переменной `c` переменную `a`
15	2	3	2	Присвоить переменной `a` переменную `b`
16	3	3	2	Присвоить переменной `b` переменную `c`
20	3	2	2	Вывод итогового состояния переменных

Самая левая колонка таблицы, озаглавленная как «Строка», содержит номера строк программы, соответствующие инструкциям, в том порядке, в котором последние выполняются. Следующие несколько колонок трассировочной таблицы, которые имеют общий заголовок «Переменные», соответствуют переменным программы. Их количество равно количеству переменных в программе. Используются они для прослеживания изменения значений переменных по ходу выполнения программы. Знак **?** в некоторых строках этих колонок соответствует неопределенному (`undefined`) значению переменной. В крайней правой колонке, «Действия», отображаются сведения о выполняемых действиях в соответствующей строке кода. ***Значения переменных в строке трассировочной таблицы приводятся до выполнения инструкции, действие которой описано в колонке «Действия».*** Инструкции, которые занимают несколько строк кода (как, например, вывода исходного и результирующего состояния переменных в **`Примере 10`**), выполняются за несколько шагов – сколько строк, столько и шагов.

Теперь, подведя небольшой итог по пройденному материалу, можно сделать вывод о том, что данный раздел является своеобразным введением в программирование на языке JavaScript. В нем были рассмотрены вопросы структуры кода скрипта, создания переменных, ввод-вывод, комментарии, отладка и т. д. То есть, все те моменты, которые в том или ином виде

встречаются в любой программе на JavaScript, и без которых не возможно создать даже самую элементарную программу.

5. Разветвляющийся вычислительный процесс

По умолчанию (т. е., если не указать специально) инструкции в программе на языке JavaScript выполняются от начала и до конца строго последовательно – т. е., реализуется линейный вычислительный процесс. Для того, чтобы в программе выполнить одну часть кода, а пропустить другую (или, наоборот), необходимо, при помощи специальных конструкций языка, выполнить ветвление (разветвление). Таких конструкций в языке программирования JavaScript всего три – две инструкции (`if` и `switch`) и один тернарный оператор (`? :`).

Инструкция `if` (русс. `если`) имеет три варианта:

1. *сокращенный*,
2. *полный* (*общий*, *стандартный*) и
3. *расширенный*.

Сокращенный вариант инструкции `if` имеет такой синтаксис:

```
if (/* Условие */) {
  // Код выполняется, если Условие истинно
}
```

Работает она следующим образом. В начале проверяется `Условие`. Если оно истинно, то выполняется блок кода, заключенный в фигурные скобки `{}`; если же нет – он пропускается. И далее выполняется следующая после `if` инструкция. Если в фигурных скобках записана всего одна инструкция, то их можно не ставить. С точки зрения синтаксиса языка блок инструкций в соответствующем месте программы воспринимается как одна (составная) инструкция – с наружи она воспринимается как одна, а внутри содержит

несколько. ***После блока инструкций, или правой закрывающейся фигурной скобки, точка с запятой не ставится!***

`Условие` в инструкции `if` – это выражение булевого (`boolean`) типа, которое возвращает значение истина (`true`) или ложь (`false`). В простейшем случае такие выражения строятся при помощи операторов отношения. Например, `if (a > 5)` – читается «если `a` больше `5`». В языке программирования JavaScript имеются такие операторы отношения:

> – больше,

< – меньше,

>= – больше или равно,

<= – меньше или равно,

== – равно (*не путать с* **=** *– оператором присваивания!*),

=== – «строгое» равно (с учетом типа),

!= – не равно.

Оператор «строгое» равно сравнивает операнды с учетом типа. Например, `'2' == 2 = true`, поскольку в начале `2` числового типа преобразуется в строковой – `'2'`, а лишь затем выполняется сравнение. `'2' === 2 = false` – потому что операнды разнотипные, а преобразование типов не выполняется.

Если же в одном выражении необходимо проверить сразу несколько условий, то в этом случае применяются логические операции. Например, `if ((a >= 5) && (b != 7))` – читается «если a больше или равно 5 и b не равно 7».

Пример 11

Написать программу, которая вводит имя пользователя и приветствует его. Если пользователь вместо имени вводит пустую строку, или нажимает кнопку **Отмена** – выдать

сообщение «Привет, Незнакомец!». Ввод исходных данных и вывод результата выполнить с помощью окон диалога.

Для этого необходимо подготовить файл script10.js такого содержания:

```
"use strict"; // Строго! Все под контролем!

var msg = 'Привет, '; // Строка сообщения
var name = prompt("Введите Ваше имя");

if (!name) name = 'Незнакомец';
msg += name + '!'; // К msg добавить имя и !

alert(msg);
```

В начале программы, при помощи инструкции var, объявляется переменная msg (сообщение) и инициализируется значением 'Привет, '. Затем создается переменная name и, при помощи функции prompt(), в нее записывается введенная пользователем информация. Далее, при помощи инструкции if, анализируется ее содержание – что же ввел пользователь? Выражение !name является Условием инструкции if. В нем, при помощи логической операции НЕ (!), проверяется содержимое переменной name. И если она пустая (''), т. е. пользователь ничего не вводил, а просто нажал кнопку **OK**, или содержит значение null – пользователь нажал кнопу **Отмена**, то ей присваивается значение 'Незнакомец'. Если же она что-то содержит, то ни чего делать не нужно – просто оставить прежнее содержание.

Рассмотренная проверка значения переменной name является частным случаем более общего правила. То есть, логическое выражение может быть построено так, что в результате вычисления оно дает значение не булевого

типа, а какого-нибудь другого, например, число или строку. В этом случае JavaScript приводит его к булевому типу на основании «правила лжи». Согласно этого правила ложью (`false`) являются следующие значения:

- число нуль – `0`,
- пустая строка – `""`,
- `null` – значение, которое, например, возвращает функция `prompt()`, когда пользователь нажал кнопку **Отмена**.
- `undefined` – значение, которое, имеет не инициализированная, вновь созданная с помощью инструкции `var`, переменная и
- `NaN` – *Not-A-Number* – указывает, что арифметическое выражение возвратило значение, не являющееся числом, например, `0 / 0`.

Все остальные значения являются истиной (`true`).

Далее в программе, при помощи инструкции:

```
msg += name + '!'; // К msg добавить имя и !
```

формируется («собирается») значение строки сообщения `msg` следующим образом. К исходному содержанию переменной `msg` (`'Привет, '`) при помощи операции «+=» конкатенируется («приклеивается») значение переменной `name` и символ «`!`».

Опять таки, этот прием тоже является одним из частных случаев. А вообще, это значит, что во многих языках программирования, и в JavaScript том числе, довольно часто встречается ситуация, кода одна и та же переменная используется и слева и справа от оператора присваивания «=». То есть, она используется и в выражении, и ей же присваивается конечный результат вычисления этого выражения. Поэтому, с целью сокращения записи, в языке программирования JavaScript помимо обычного оператора присваивания «=» реализованы несколько других, двухсимвольных, *операторов-сокращений* (*составных операторов, сложного*

присваивания), позволяющих объединять присваивание с некоторой другой операцией. Например, оператор «+=» выполняет сложение и присваивание. Следующие выражения эквивалентны:

```
x += 2;
x = x + 2;
```

и означают они, что к текущему значению переменной `x` необходимо прибавить `2`, и результат присвоить той же переменной `x`, т. е. оба они увеличивают значение переменной `x` на `2`. Оператор «+=» работает и с числами, и со строками: если операнды числовые, он выполняет сложение и присваивание, а если строковые – конкатенацию и присваивание. Из подобных ему операторов можно еще назвать «-=», «*=», «/=», «%=» и др.

В языке программирования JavaScript полный вариант инструкции `if` имеет такой синтаксис:

```
if (/* Условие */) {
  // Код выполняется, если Условие истинно
} else {
  // Код выполняется, если Условие ложно
}
```

А его *семантика*, т. е. правила выполнения таковы. В начале проверяется условие. Если оно истинно, то выполняется блок инструкций после ключевого слова `if` и пропускается после ключевого слова `else` (русс. `иначе`), а если наоборот – ложно – то пропускается после `if` и выполняется после `else`. То есть, полная форма инструкции `if` применяется, когда необходимо обеспечить выполнение одного из двух участков кода. Продемонстрируем это на **Примере 12**.

Пример 12

Изменить предыдущую программу (**Пример 11**) так, чтобы для проверки введенного пользователем значения использовалась полная форма инструкции if.

Для этого создадим сценарий (файл script10.js) такого содержания:

```
"use strict"; // Строго! Все под контролем!

var msg = 'Привет, '; // Строка сообщения
var name = prompt("Введите Ваше имя");

if (name) { // Если name что-то содержит
  msg += name + '!';
} else {    // Если name недопустимое
  msg += 'Незнакомец' + '!';
}

alert(msg);
```

Он отличается от предыдущего способом проверки введенного пользователем в переменную name значения и способом формирования результирующего сообщения – переменной msg. Если введенное значение допустимое – переменная name что-то содержит – то к исходному значению переменной msg конкатенируется это значение. Если же не допустимое – то к ней «прибавляется» строковая константа 'Незнакомец'. В любом случае в конец переменной msg, при помощи операции конкатенации, дописывается символ восклицательного знака «!». Сформированное таким образом значение переменной msg, при помощи функции alert(), выводится в окно диалога.

В языке программирования JavaScript реализация логических операций имеет некоторые особенности. Одна из них – «сокращенный алгоритм» выполнения таких операций. Это значит, что логическое выражение анализируется слева направо, и такой анализ не всегда доходит до конца, а завершается, как только становится известен его результат. Например, `a || b` – если `a` – истина, то и все выражение истинно и дальше ни чего не анализируется. Наибольший же выигрыш получается на сложных выражениях – чем оно сложнее, тем больше выигрыш. Следующая особенность состоит в том, что логические `И/ИЛИ` возвращают не булево значение, а один из операндов, определивший значение выражения. Например, `a = b || 'просто текст'` – переменная `a` получит значение переменной `b`, если последняя содержит не пустое значение, а если пустое (`''`) – `'просто текст'`. Если в этом выражении поменять местами операнды (`a = 'просто текст' || b`) переменная `a` всегда будет иметь значение `'просто текст'`, поскольку значение выражения – `true` (`'просто текст'` – не пустая строка) – известно сразу.

Использование этих особенностей языка JavaScript для формирования той же, что и ранее, строки сообщения рассмотрим на **Примере 13**.

Пример 13

Изменить предыдущую программу (**Пример 12**) так, чтобы для формирования результирующего сообщения вместо инструкции `if` была использована логическая операция ИЛИ (`||`).

Решение поставленной задачи заключается в следующем. При помощи операции `ИЛИ` записывается логическое выражение. Левый операнд этого выражения – возвращаемое функцией `prompt()` значения, а правый – константа строкового типа `'Незнакомец'`. Если введенное при помощи

функции prompt() значение допустимое, т. е. преобразуется к логическому значению истина (не пустая строка и не null), то оно присваивается переменной name. Если же это значение не допустимое, то переменной name будет присвоен правый операнд выражения – строковая константа 'Незнакомец'. Реализация этого алгоритма на JavaScript в виде файла script11.js приведена далее:

```
"use strict"; // Строго! Все под контролем!

var msg = 'Привет, '; // Строка сообщения
var name = prompt("Введите Ваше имя")
           || 'Незнакомец';

msg += name + '!';

alert(msg);
```

В принципе, эту программу можно «сократить» и до одной инструкции, исключив даже всякие переменные. Рассмотрим этот вариант на **Примере 14**.

Пример 14

Изменить предыдущую программу (**Пример 13**) так, чтобы она была реализована при помощи одной инструкции.

Для этого, просто, необходимо создать фай script12.js такого содержания:

```
alert('Привет, '
      + (prompt("Введите Ваше имя")
      || 'Незнакомец') + '!');
```

Особенностью этого скрипта является наличие множества скобок и их порядок. Это связано с тем, что различные операции имеют различный приоритет. В этом выражении операция конкатенация «+» имеет приоритет выше, чем операция логическое ИЛИ. И поэтому, чтобы она выполнилась первой, ее, вместе с операндами, необходимо взять в скобки.

Этот пример еще раз продемонстрировал также, что одну и ту же программу можно написать «сто и одним» способом. А какой из них самый лучший? Тот, который наиболее очевиден Вам. И чем больше таких вариантов Вы знаете, тем более быстро и качественно сможете создавать программы.

Если же необходимо обеспечить разветвление кода не на две, а на большее число веток, то применяется расширенный вариант инструкции `if`. Он имеет такой синтаксис:

```
if (/* Условие_1 */) {
  // Код выполняется, если Условие_1 истинно
} else if (/* Условие_2 */) {
  // Код выполняется, если Условие_2 истинно



} else if (/* Условие_n */) {
  // Код выполняется, если Условие_n истинно
} else {
  // Код выполняется, если все n Условий ложны
}
```

В нем выполняется лишь тот участок кода, «чье» `Условие` истинно – все остальные пропускаются. Если ни одно из `Условий` не выполняется, и имеется

ветвь `else`, то выполняется она. Если же ветви `else` нет, то ничего и не выполняется – управление получает следующая после `if` инструкция.

Особенности применения расширенного варианта инструкции `if` в программах JavaScript рассмотрим на **Примере 15**.

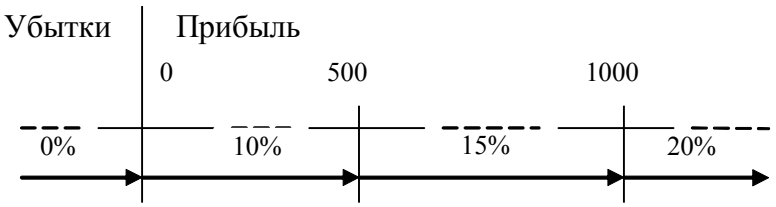
Пример 15

Написать программу начисления налога на прибыль в соответствии со следующей таблицей:

Прибыль	Ставка налога
[денежная единица]	[%]
до 500	10
от 500 до 1000	15
1000 и более	20

Ввод исходных данных и вывод результата выполнить при помощи окон диалога.

Графически шкалу начисления налога, в зависимости от полученной прибыли, можно представить следующим образом:



Поскольку на убытки, или отрицательную прибыль, налог не начисляется, то в программе необходимо предусмотреть обработку и такой ситуации. В ней также необходимо предусмотреть две переменные числового типа для хранения:

1. суммы прибыли и
2. суммы начисляемого налога.

Назовем их `profit` и `tax`, соответственно. Тогда процедуру начисления налога на прибыль можно записать следующим образом (фай `script13.js`):

```
"use strict"; // Строго! Все под контролем!

var tax;      // Сумма налога
var profit = +prompt("Введите сумму прибыли");

// Начисление налога
if (profit <= 0) {
  tax = 0;
} else if (profit < 500) {
  tax = profit * 0.1;
} else if (profit < 1000) {
  tax = profit * 0.15;
} else {
  tax = profit * 0.2;
}

// Вывод результата
alert("Налог равен: " + tax);
```

Отформатированная таким образом программа наиболее более точно отражает (по крайней мере, визуально) суть поставленной задачи, а именно – начисление налога на прибыль в зависимости от ее величины и процентной ставки.

Альтернативой расширенной формы инструкции `if` является инструкция `switch` (русс. `переключатель`), которая также обеспечивает множественное ветвление:

```
switch (/* Выражение */) {
  case /* Значение_1 */ :
    // Код выполняется, если Выражение === Условие_1
    break;
  case /* Значение_2 */ :
    // Код выполняется, если Выражение === Условие_2
    break;



  default:
    // Код выполняется, если Выражение != Условиям
}
```

В ней значение Выражения последовательно (сверху низ) «строго» сравнивается со Значением после каждого ключевого слова case (русс. случай). И как только такое равенство найдено, начинается выполнение кода после соответствующего case. Для того, чтобы выполнение кода не «проваливалось» до самого конца инструкции switch, используется инструкция break. Она обеспечивает передачу управления за пределы инструкции switch, т. е. обеспечивается выполнение лишь одной ветки алгоритма. Ключевое слово case, Значение и символ двоеточия «:» в конце – это, по сути дела, метка, на которую передается управление в случае (англ. case), если Выражение === Значение. Этим инструкция switch отличается от инструкции if, в которой управление ни когда и ни куда не «проваливается» – выполняется (или, не выполняется) только один блок кода. Если же значение Выражения не совпало ни с одним из Значений case, и имеется ветвь default (русс. типовое значение, значение по умолчанию) – выполняется она. Ставить break в ветке default не обязательно – управление и так будет передано за пределы switch.

Специфику применения инструкции `switch` в программах на языке JavaScript рассмотрим на **Примере 16**.

Пример 16

Написать программу, которая спрашивает у пользователя число ворон на ветке (от 0 до 5), а выводит сообщение "На ветке … ворон" (слово ворон вывести в правильном падеже). Ввод исходных данных и вывод результата выполнить при помощи окон диалога.

Для этого необходимо создать скрипт `script14.js` такого содержания:

```
"use strict"; // Строго! Все под контролем!

var crows = prompt("Сколько ворон на ветке?");
var msg = 'На ветке ';

switch (crows) {
  case '0':
    msg += 'нет ни одной вороны';
    break;
  case '1':
    msg += 'одна ворона';
    break;
  case '2':
    msg += 'две вороны';
    break;
  case '3':
    msg += 'три вороны';
    break;
```

```
  case '4':
    msg += 'четыре вороны';
    break;
  case '5':
    msg += 'пять ворон';
    break;
  default:
    msg += 'неизвестное число ворон';
}

alert(msg);
```

Особенностью данного скрипта является использование инструкций `switch` и `break`. Также в нем введенная пользователем информация (значение переменной `crows` – количество ворон) не преобразуется к числовому типу, а остается текстовым. Поэтому в ветвях `case` применяются не числовые, а текстовые константы (например, `case '2':`) – в инструкции `switch` всегда используется только «строгое» (с учетом типа) сравнение.

Еще одним способом выполнения ветвления в программах на языке JavaScript является использование тернарного оператора, который имеет такой синтаксис:

```
/* Условие */ ? /* Значение_1 */ : /* Значение_2 */;
```

И работает следующим образом. В начале проверяется `Условие`. Если оно истинно, то оператор возвращает `Значение_1`, а если ложно – `Значение_2`. Он используется, когда необходимо не просто выполнить сравнение, а вернуть одно из двух значений, в зависимости от такого сравнения. Например, «присвоить переменной `a` значение большей из двух переменных – `b` или `c`» –

`a = (b > c) ? b : c;`. Это же можно реализовать и при помощи инструкции `if` – но более длинно:

```
if (b > c)
  a = b;
else
  a = c;
```

В качестве демонстрации возможностей тернарного оператора в программах на JavaScript рассмотрим **Примере 17**.

Пример 17

Написать программу, которая:

1. вводит возраст посетителя,
2. записывает во внутреннюю переменную **access** значение **true**, если его возраст больше или равен 18 лет, и **false** – если меньше,
3. выводит значение переменной **access**.

Ввод исходных данных, а также вывод результата выполнить при помощи окон диалога.

Поставленная задача решается при помощи скрипта `script15.js` такого содержания:

```
"use strict"; // Строго! Все под контролем!

var access;
var age = prompt("Сколько Вам лет?");

access = (age >= 18) ? true : false;
```

```
alert(access);
```

Единственной его особенностью является использование тернарного оператора (`? :`). В таком контексте программа получилась более наглядной и изящной, чем в случае применения традиционной инструкции `if`. Поэтому, в дальнейшем, этот код также может быть взят за основу при построении более совершенных и надежных систем контроля и ограничения доступа к определенным ресурсам.

6. Циклический вычислительный процесс

При создании программ довольно часто возникает необходимость многократного выполнения некоторого участка кода. Для ее реализации практически во всех языках программирования имеются конструкции под названием *ц и к л ы* . Использование циклов позволяет существенно упростить и сократить длину программного кода.

Всякий цикл состоит из двух частей:

- *з а г о л о в о к* и
- *т е л о* .

Тело цикла представляет собой участок кода, который необходимо выполнить несколько раз. Заголовок же управляет процессом выполнения тела цикла. Однократное выполнение тела цикла называется *и т е р а ц и е й* .

В программировании существует два типа циклов:

- *о п р е д е л е н н ы е* – когда число повторений тела цикла известно к началу его выполнения, и
- *н е о п р е д е л е н н ы е* – когда число повторений тела цикла заранее неизвестно.

Определенные циклы в языке программирования JavaScript реализуются при помощи инструкций `for` и `for-in`, а неопределенные – `while` и `do-while`.

Цикл `while` (англ. `пока`) имеет такой синтаксис:

```
while (/* Условие */) {
  // Тело цикла
}
```

Семантика цикла while такова: «до тех пор, пока истинно Условие, выполнять Тело цикла». Если при первой же проверке Условие будет иметь значение ложь – Тело цикла не выполнится ни разу.

Для тог, чтобы Тело цикла выполнилось хотя бы один раз, проверку Условия ставят после Тела цикла – и получают цикл do-while, который имеет такой синтаксис:

```
do {
  // Тело цикла
} while (/* Условие */);
```

Цикл, записанный таким образом, сначала выполняет Тело цикла, а затем проверяет Условие.

Специфику применения неопределенных циклов do-while в программах на языке JavaScript рассмотрим на **Примере 18**.

Пример 18

Написать программу, которая будет спрашивать имя пользователя до тех пор, пока он не введет допустимую строку. А затем – поздоровается с ним. Ввод исходных данных и вывод результата выполнить с помощью окон диалога.

Поскольку заранее не известно, сколько раз пользователь будет пытаться вводить свое имя, то необходимо использовать один из циклов: do или do-

while. То есть, «*запрос в цикле необходимо выдавать до тех пор, пока пользователь не введет допустимое имя*». Из этой формулировки следует, что «*выдать запрос необходимо хотя бы один раз, а лишь затем – его проверить*» – «сам напрашивается» цикл do-while, в котором проверка условия осуществляется после выполнения тела цикла. Тогда реализация этого алгоритма на JavaScript должна выглядеть следующим образом (файл script16.js):

```
"use strict"; // Строго! Все под контролем!

var userName; // Имя пользователя

do {
  userName = prompt("Введите Ваше имя");
} while(!userName);

alert('Привет, ' + userName + '!');
```

При создании программ довольно часто возникает необходимость изменять значение переменной на единицу. Например, в циклах при обработке всех элементов некоторой последовательности. Поэтому во многих языках программирования, и в JavaScript в том числе, реализованы две специальные *унарные* операции:

++ – *инкремент* и

-- – *декремент*,

которые увеличивают или, соответственно, уменьшают значение операнда на единицу. Следующие выражения эквивалентны между собой:

```
i = i + 1; i += 1; i++;  // Увеличить i на 1
i = i - 1; i -= 1; i--;  // Уменьшить i на 1
```

Операции инкремент и декремент могут применяться только к операндам, расположенным в оперативной памяти компьютера, т. е. к переменным, но не к выражениям!

Операторы инкремента и декремента могут использоваться в *п р е ф и к с н о й* или *п о с т ф и к с н о й* формах:

```
++i; --i; // Префиксная форма
i++; i--; // Постфиксная форма
```

При этом, если знак операции стоит перед операндом (префиксная форма), то сначала производится операция – инкремент или декремент – а потом возвращается значение операнда, уже, естественно, измененное. Если же знак оператора стоит после операнда (постфиксная форма), то сначала возвращается неизменное значение операнда, а потом производится операция – изменение операнда.

Применение неопределенного цикла `while` и операции декремента в языке программирования JavaScript рассмотрим на **Примере 19**.

Пример 19

Написать программу вычисления факториала числа n – произведения всех натуральных чисел от 1 до n включительно: $n! = 1 \cdot 2 \cdot \ldots \cdot n = \prod_{i=1}^{n} i$. Например: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$. По определению полагают 0! = 1. Ввод исходных данных и вывод результата выполнить с помощью окон диалога.

Одна из реализаций алгоритма вычисления факториала заключается в том, что в цикле исходное число на каждой итерации умножается на текущее значение факториала, а затем – уменьшается на 1. Процесс завершается, как

только исходное число станет равным 0. Сама же реализация этого алгоритма, в виде файла программы на JavaScript (script17.js), будет иметь такое содержание:

```
"use strict"; // Строго! Все под контролем!

// Объявление переменных
var n,              // Исходное число n
    nFactorial = 1, // Факториал числа n
    tmp;            // Временная переменная

do {                // Ввод и проверка исходных данных
  n = +prompt("Введите число n, больше нуля");
} while(!n || (n < 0));

n -= n % 1;         // n в целое
tmp = n;            // Сохранить исходное n
while(n) {          // Вычисление факториала
  nFactorial *= n--;
}

// Вывод результата
alert(tmp + '! = ' + nFactorial);
```

Код программы получился не совсем простым, очевидным и коротким. Поэтому, далее более детально рассмотрим его особенности, которые заключаются в следующем:

- помимо переменных для самого числа (n) и факториала (nFactorial) была введена дополнительная переменная tmp для сохранения исходного числа n, поскольку оно в процессе вычисления уменьшается до 0;

- для проверки того, что пользователь ввел именно число больше нуля, при помощи цикла `do-while`, осуществляется его контроль на допустимость (практически такой же, как и в предыдущем **Примере 18**);
- если пользователь ввел нецелое число, то необходимо преобразовать его в целое, отбросив дробные разряды; для этого с помощью операции остаток от целочисленного деления «`%`» числа `n` на `1` вычисляется дробная часть `(n % 1)` и вычитается из самого числа `n - (n % 1)`; результат снова записывается в переменную `n = n - (n % 1);` или более компактно: `n -= n % 1;`.
- при вычислении факториала использованы два оператора: умножение с присваиванием «`*=`» и постфиксный инкременте «`--`», который на каждой итерации уменьшает значение `n` на `1` после его «использования»; цикл заканчивается, как только `n` становится равным `0`, т. е. реализован алгоритм `nFactorial = n * (n – 1) * (n – 2)*…* 1;` переменная, в которой в цикле «накапливается» произведение (`nFactorial`), до начала цикла проинициализирована `1` – умножение любого числа на `1` не изменяет его (если бы накапливалась сумма, переменную необходимо было бы инициализировать `0` – сложение любого числа с `0` не изменяет его).

В программировании, все-таки, более часто используются определенные циклы. Определенный цикл `for` («`для` всех значений последовательности») в языке программирования JavaScript имеет следующий синтаксис:

```
for (/* Инициализация */; /* Условие */; /* Шаг */) {
  // Тело цикла
}
```

В нем всегда имеется некоторая переменная, которая управляет количеством повторений тела цикла. Она так и называется – *переменная цикла* или *счетчик*. После каждой итерации (выполнения тела цикла) она или

увеличивается на величину шага – если цикл повторяется «от меньшего к большему» – или уменьшается – если наоборот. Такие переменные, как правило, называются `i`, `j` или `k`.

Цикл `for` «работает» следующим образом:

1. В начале `Инициализируется` переменная цикла (здесь также могут быть присвоены некоторые начальные значения и другим переменным).
2. Затем проверяется `Условие`. Если оно истинно, то выполняется `Тело цикла`, а если ложно – цикл завершается, и управление передается на следующую после цикла инструкцию.
3. После выполнения `Тела цикла` переменная цикла получает приращение (положительное или отрицательное) на величину `Шага`, и снова проверяется `Условие`.

Любая часть инструкции `for` может быть пропущена (`Инициализация`, `Условие` или `Шаг`). При этом сами точки с запятой «`;`» обязательно должны присутствовать, иначе будет ошибка синтаксиса. Например, если вообще убрать все компоненты, то получится бесконечный цикл:

```
for (;;) {
  // Будет выполняться вечно
}
```

Бесконечные циклы иногда применяются в программировании, но чаще всего свидетельствуют об ошибках в логике работы программ.

Продемонстрируем работу определенного цикла `for` языка программирования JavaScript на **`Примере 19`**.

`Пример 19`

`Написать программу, которая на Web-страницу выводит таблицу квадратов чисел от 0 до 9.`

Для этого создадим файл Web-страницы `for1.html` такого содержания:

```
<html>
  <head>
    <title>JavaScript</title>
    <meta charset=utf-8>
  </head>
  <body>
    <h1>Таблица квадратов</h1>
    <script>
      "use strict"; // Строго! Все под контролем!

      document.write("<h2>Цикл for</h2>");

      for (var i = 0; i < 10; i++) {
        document.write(i + ' * ' + i + ' = ' + i * i +
"<br>");
      }
    </script>
  </body>
</html>
```

Особенность данного кода является то, что для вывода информации на Web-страницу была задействована еще одна возможность JavaScript, а именно метод `write()` объекта `document` – `document.write()`. Метод `write()` служит для вывода на HTML-страницу данных, которые были переданы ему как параметр. Он может выводить не только текст, но и теги. То есть, с помощью метода `write()` объекта `document` можно менять содержимое Web-страницы, что и было продемонстрировано в данном скрипте дважды.

Первый раз – для вывода текста «Цикл for» отформатированного заголовком второго уровня (<h2>) и второй раз – в цикле, для формирования строк таблицы.

В цикле for, в качестве его параметра, была использована переменная i. Она последовательно «пробегает» значения от 0 и до 9, обеспечивая, тем самым, десятикратное выполнения тела цикла – инструкции вывода очередной строки таблицы:

```
document.write(i + ' * ' + i + ' = ' + i * i + "<br>");
```

Сама же строка вывода – это параметр метода write(), который формируется из отдельных составляющих при помощи операции конкатенации.

Для придания большей гибкости при организации циклов в программах на языке JavaScript применяются две специальные инструкции: break (русс. прервать) и continue (русс. продолжить). Инструкция continue прерывает выполнение текущей итерации и переходит к следующей, т. е. все инструкции после continue и до конца тела цикла пропускаются. Сам цикл при этом не прекращается, а пропускается только текущая итерация. Для немедленного прекращения цикла служит инструкция break. Она немедленно прекращает выполнение текущей итерации и передает управление на следующую после цикла инструкцию.

Использование инструкции continue рассмотрим на следующем **Примере 20**.

Пример 20

Измените программу **Пример 19** так, чтобы при выводе таблицы квадратов она пропускала числа 2 и 5.

Для этого необходимо отредактировать файл `for1.html`, добавив в тело цикла, перед инструкцией вывода очередной строки, такой код:

```
if ((i == 2) || (i == 5)) continue;
```

При выполнении этого условия очередная строка таблицы выводиться не будет – управление циклом будет передано на его начало.

Досрочное завершение цикла в программах на JavaScript при помощи инструкции `break` рассмотрим на **Примере 21**.

Пример 21

Измените программу **Пример 20** так, чтобы после строки с числом 7 она заканчивала вывод.

Для этого в файле `for1.html` необходимо отредактировать цикл `for`, добавив вызов инструкции `break` по условию. В результате двух последних изменений этот фрагмент кода должен принять такое содержание:

```
for (var i = 0; i < 10; i++) {
  if ((i == 2) || (i == 5)) continue;
  document.write(i + ' * ' + i + ' = ' + i * i + "<br>");
  if (i == 7) break;
}
```

Циклы также можно вкладывать друг в друга. Это необходимо, например, при работе с таблицами, у которых два измерения: номер строки и номер столбика. При этом, если вкладываются друг в друга циклы `for`, то переменные циклов должны быть разными, например, `i` и `j` и внутренний цикл должен полностью укладываться во внешний.

Следующий **Пример 22** демонстрирует особенности применения вложенных циклов в программах на языке JavaScript.

Пример 22

Напишите программу, которая создает на Web-странице таблицу умножения 9 × 9.

Типичным способом решения этой (и аналогичных ей) задачи является использование вложенных циклов `for`. Внешний цикл обеспечивает последовательный «перебор» всех строк, а внутренний – столбиков (ячеек). Вывод на Web-страницу тегов формирования элементов таблицы и другой необходимой информации выполняется при помощи метода `document.write()`. Реализация этого алгоритма в виде кода на JavaScript приведена далее. Соответствующий файл (`for2.html`) должен иметь такое содержание:

```
<html>
  <head>
    <title>JavaScript</title>
    <meta charset=utf-8>
  </head>
  <body>
    <h1>Таблица умножения</h1>
    <script>
      "use strict"; // Строго! Все под контролем!

      var n = 9;    // Размер таблицы

      document.write("<h2>Вложенные циклы</h2>");
```

```
      document.write("<table border=1>");// Начало
таблицы
      for (var i = 1; i <= n; i++) {      // Цикл по
строкам
        document.write("<tr>");           // Начало строки
        for (var j = 1; j <= n; j++) {    // Цикл по
ячейкам
          document.write("<td>");         // Начало ячейки
          document.write(i * j);          // Содержание
          document.write("</td>");        // Конец ячейки
        }
        document.write("</tr>");          // Конец строки
      }
      document.write("</table>");         // Конец таблицы
    </script>
  </body>
</html>
```

Более детально функции, которые выполняет каждая строка кода, объясняют их однострочные комментарии.

Таблица, формируемая при помощи нашего скрипта, получилась довольно однообразной и не совсем наглядной. Для придания ей более привлекательного внешнего вида рассмотрим следующий **Пример 23**.

Пример 23

Усовершенствуйте программу из предыдущего **Примера 22** так, чтобы первая строчка и первый столбик были зеленого цвета.

Для этого необходимо в инструкции формирования начала ячейки первой строки и первого столбца добавить атрибут цвета фона. То есть, заменить единственную инструкцию формирования начала ячейки инструкцией `if`, в которой проверять номер строки и столбца. И если хотя бы один из них равен `1`, то в инструкцию формирования начала ячейки добавить атрибут `bgcolor=`, а если нет – использовать «старую» инструкцию. Поэтому, такой фрагмент кода на JavaScript должен выглядеть следующим образом:

```
if ((i == 1) || (j == 1))       // Начало ячейки
  document.write("<td bgcolor = '#00ff00'>")
else
  document.write("<td>");
```

Существует также специальная конструкция определенного цикла – `for-in` – для перебора *свойств объекта*. Подробности ее применения в программах на JavaScript рассматриваются в теме посвященной объектам.

ЛЕКЦИЯ № 10

ПЕРСПЕКТИВЫ РАЗВИТИЯ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

План

1. Качество и достоверность информации
2. Основные направления развития информационных технологий
3. Облачные технологии
4. IT-аутсорсинг
5. IT-фриланс
6. Миграция на свободное программное обеспечение

1. Качество и достоверность информации

Среди множества проблем, стоящих перед современным обществом, особое место занимает качество информации. Всем известна поговорка: «Кто владеет информацией – тот владеет миром». Однако каждый также может вспомнить далеко не единичные случаи, когда даже в бытовых, часто повторяющихся ситуациях принимались не самые эффективные решения. Так происходит потому, что эти решения принимались на основании недостоверной информации (эмоциональная составляющая здесь не учитывается).

Уже в месте возникновения и регистрации информация может содержать ошибки, связанные с точностью ее измерения и аккуратностью фиксации на носителях. Далее она передается по различным каналам связи, где также может быть подвержена искажению в результате воздействия различного рода помех. И, наконец, в местах обработки в нее также могут быть «добавлены» ошибки. Успешно решать эту проблему, т. е. качественно «отфильтровывать» ошибки, возникающие на всех этапах обработки данных, и в результате получать достоверную выходную информацию можно только на базе передовых информационных технологий. Поэтому во всем мире проблеме эффективного использования информационных технологий уделяется самое пристальное

внимание. И как весьма существенное обстоятельство следует отметить тот факт, что за последние несколько лет в Украине был принят целый ряд законодательных актов, которые регламентируют использование информационных технологий в органах государственной власти.

2. Основные направления развития информационных технологий

Основными направлениями развития информационных технологий на сегодняшний день, которые тесно переплетаются, дополняют друг друга и, в конечно итоге, самым существенным образом влияют на стратегию и тактику их использования во всех областях жизни общества, являются:

- облачные технологии,
- IT-аутсорсинг,
- IT-фриланс,
- миграция на свободное программное обеспечение.

3. Облачные технологии

Облачные технологии (облачные вычисления, англ. cloud computing) – это одна из передовых IT-парадигм, в которой вычислительные ресурсы предоставляются пользователю как Интернет-сервисы. Пользователь имеет доступ к собственным данным, но не заботится о вычислительной инфраструктуре, с которым он работает. При этом предоставляется возможность значительно оптимизировать расходы на инфраструктуру информационных технологий (в краткосрочном и среднесрочном планах), а также гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности (англ. elastic computing) облачных услуг.

Термин «Облако» используется как метафора, основанная на изображении Интернета на схемах компьютерных сетей, или как образ сложной инфраструктуры, за которой от клиента скрываются все технические детали обработки информации. Другими словами под облаком понимается некий

крупный дата-центр, где совершаются все вычислительные операции. При этом автоматически снимаются все проблемы, связанные с производительностью компьютеров, а также количеством свободного пространства на их жестких дисках.

Основными моделями обслуживания пользователей в облачных технологиях являются:

- инфраструктура как услуга (англ. IaaS – Infrastructure-as-a-Service),
- рабочее место как услуга (англ. WaaS– Workplace-as-a-Service),
- данные как услуга (англ. DaaS – Data-as-a-Service),
- программное обеспечение как услуга (англ. SaaS – Software-as-a-Service),
- платформа как услуга (англ. PaaS – Platform-as-a-Service)

и некоторые другие технологические тенденции, общим в которых является уверенность в том, что сеть Интернет всегда в состоянии удовлетворить потребности пользователей в обработке данных.

Одним из наиболее интересных направлений развития облачных технологий является электронное правительство, концепция развития которого в Украине на период до 2015 года была одобрена на заседании Кабинета министров 13 декабря 2010 года [55]. Реализовывать концепцию планировалось в три этапа. На первом этапе (до 2012 г.) необходимо было разработать нормативно-правовую и нормативно-техническую базы, необходимые для предоставления административных услуг в электронном виде. В рамках второго этапа планировалось, что до 2014 года в электронной форме будут предоставляться услуги во всех сферах общественной жизни. На третьем этапе, который продлился до 2015 г., предполагалось создать единую информационно-телекоммуникационную инфраструктуру органов государственной власти и местного самоуправления. Практическим результатом реализации этой концепции стало то, что уже сегодня отчеты во все контролирующие органы Украины принимаются только в электронном виде через Интернет.

4. IT-аутсорсинг

IT-аутсорсинг (англ. ITO – IT-outsourcing, outsourcing – вынос вовне) – это способ оптимизации деятельности предприятия за счет частичного или полного делегирования функций, связанных с IT-технологиями, внешним специализированным компаниям, для которых выполнение подобных работ является профильным направлением деятельности. Такие компании имеют соответствующий штат специалистов необходимой квалификации.

Основными сферами использования IT-аутсорсинга являются:

- Обслуживание информационных систем предприятия (или абонентское обслуживание), которое включает в себя такие виды услуг, как настройка и обновление аппаратной и программной части системы, антивирусная профилактика и т. д.
- Аутсорсинг дата-центров или ЦОД (Центров Обработки Данных).
- Внешнее размещение информационных систем, когда DaaS-аутсорсер не только предоставляет физическое оборудование для размещения информационных систем, но и обеспечивает их установку, поддержку и обновление.
- Разработка программного обеспечения.

5. IT-фриланс

IT-фриланс (англ. freelance – свободное копье) имеет много общего с IT-аутсорсингом – в обоих случаях для нанимающей их компании они являются сторонними исполнителями. Только в случае аутсорсинга исполнителем выступает некоторая компания, а в случае фриланса – отдельный работник – фрилансер («вольный художник», внештатный работник). Т. е. человек выполняет работу без заключения долговременного договора с работодателем, нанимается только для выполнения определенного перечня работ, но все же для него – это систематическая работа. Будучи вне постоянного штата какой-либо компании, фрилансер может одновременно выполнять заказы для разных клиентов.

Со временем фрилансер вырастает и понимает, что вполне может создать свою фирму для предоставления услуг аутсорсинга. Это дает ему и его команде заметный выигрыш, нежели одиночная работа, особенно в крупных IT-проектах. Кроме того, в условиях кадрового голода в IT-отрасли зачастую компании аутсорсинга сами привлекают фрилансеров.

Согласно статистике в 2012 году общая сумма контрактов, реализованных фрилансерами Украины, составила 34,3 млн. долларов, из них 32 млн. дол. приходится на IT-специалистов. Эти объемы позволили Украине занять 4-е место в рейтинге стран по уровню доходов фрилансеров. Первое место занимает Индия (150,9 млн. дол.), второе – США (150,2 млн. дол.) и третье – Пакистан (35,6 млн. дол.) [58].

В 2014 году в Харькове стартовало создание масштабного IT-парка [47]. Масштабность проекта состоит в том, что под одной крышей будут сосредоточены множество небольших IT-компаний, а также независимых фрилансеров, работающих в этой индустрии. Здание будущего комплекса для программистов площадью 14 тыс. кв. м. находится возле станции метро «Московский проспект». Там будут размещаться рабочие зоны, зоны для отдыха, общения и обучения, для того чтобы специалисты могли работать, отдыхать и обмениваться знаниями, опытом, мыслями.

6. Миграция на свободное программное обеспечение

В последнее время во всем мире наметилась устойчивая и необратимая тенденция постепенного перехода (миграции) на свободное программное обеспечение (СПО, англ. free software). Украина в этом процессе занимает одно из лидирующих мест. Так, одним из первых положительных опытов использования свободного программного обеспечения в государственном секторе были выборы Президента Украины в 1999 году [49]. Тогда для автоматизации избирательных участков было задействовано 225 компьютеров под управлением локализованной в Украине свободной операционной системы KSI Linux. Никаких проприетарных программных продуктов на них не было

установлено – использовалось только свободно распространяемое ПО. Необходимо особо подчеркнуть, что за время работы выборов не было зафиксировано ни единого сбоя ПО!

Парк компьютеров в украинских госорганах на сегодняшний день составляет около 250 тыс. единиц, значительная часть из которых приобретена с ПО, на копирование или модификацию которого наложены ограничения. Чтобы легализировать уже существующие программы и докупить новое лицензионное ПО, необходимо выделить значительные бюджетные средства. В частности, дальнейшее использование проприетарного ПО в органах государственной власти означает необходимость каждые 4-5 лет тратить от 400 до 1500 млн. грн.

Поэтому, с целью экономии бюджетных средств, Кабинет министров Украины своим постановлением № 1269 от 30 ноября 2011 г. утвердил Государственную целевую научно-техническую программу постепенного перехода органов государственной власти на программное обеспечение с открытым кодом на 2012 ÷ 2015 годы [52]. При этом экономия средств от перехода госаппарата на открытое ПО может достигнуть 87 %.

23 января 2012 года группа депутатов зарегистрировала под номером 9745 проект Закона Украины «О внесении изменений в Закон Украины «О Национальной программе информатизации» об использовании открытого программного обеспечения в органах государственной власти, органах местного самоуправления, государственных учреждениях, государственных предприятиях и учебных заведениях государственной и коммунальной формы собственности» [53].

Этот законопроект вызывает повышенный интерес на фоне сложившейся за последнее время ситуации в сфере информатизации в Украине. В частности там говорится, что при выборе поставщиков программного обеспечения необходимо отдавать предпочтение программным продуктам отечественных разработчиков, а также программным продуктам с открытым кодом.

Отдельно в проекте Закона сформулированы требования по использованию программного обеспечения в учебных заведениях. В частности, там говорится:

1. Предметом исследования и изучения в учебных заведениях государственной и коммунальной формы собственности должны быть общие принципы, методы, механизмы и логика функционирования современного программного обеспечения и информационных технологий на примерах открытого и закрытого программного обеспечения.
2. Учебные заведения государственной и коммунальной формы собственности обязаны использовать в учебном процессе открытые технические стандарты и программное обеспечение, которое никоим образом не требует дополнительной оплаты со стороны ученика.
3. Учебные заведения государственной и коммунальной формы собственности могут использовать закрытое программное обеспечение в исследовательских целях в случаях, когда проведение исследования непосредственно связано с использованием исключительно конкретной программы.

В рекомендациях парламентських слушаний № 1565-VII от 03.07.2014 г. «Законодательное обеспечение развития информационного общества в Украине» еще раз подчеркивается необходимость перехода на свободное программное обеспечение и усиление контроля его выполнения [54].

Миграция (постепенный переход) на свободное программное обеспечение представляет собой многоплановый и не единоразовый процесс. Он включает такие аспекты, как организационные, правовые, технические и д. р. Технический переход, как один из основных компонентов, общей задачи перехода на СПО, может быть выполнен несколькими различными способами или в несколько этапов. Основными из них являются такие:

1. Установка Linux на «чистый» компьютер. Применяется в «экстренных» ситуациях и как завершающая фаза процесса перехода на СПО.

2. Множественная загрузка операционных систем. При этом существенным является последовательность их установки. Поскольку не все версии операционной системы Windows легко устанавливаются после Linux, то более предпочтительной является последовательность, когда после Windows ставится Linux – Linux «относится» более «дружественно» к проприетарному ПО, чем оно к нему.
3. Использование виртуальных машин – специальных программ, которые на одной операционной системе позволяют эмулировать работу другой операционной системы, например, Oracle VM VirtualBox.
4. Использование специальной программы WINE, которая позволяет запускать Windows-приложения под Linux.
5. Многие дистрибутивы Linux поставляются в виде LiveCD или LiveDVD. Т. е. можно выполнить загрузку компьютера с такого диска и опробовать работу операционной системы, а затем уже решать, стоит ее устанавливать, или нет.
6. Использование кроссплатформенных программ, т. е. программ которые могут выполняться под управлением различных операционных систем, например, OpenOffice или LibreOffice – свободные аналоги проприетарного Microsoft Office. Достаточно освоить работу с такой программой под управлением своей привычной операционной системы, например, Microsoft Windows, а затем с ней уже будет более привычно работать и под Linux.

Как показывает мировой опыт, практический переход на СПО более предпочтительно начинать с последнего этапа, а заканчивать – первым. Такая последовательность позволяет наиболее плавно и безболезненно «мигрировать» на СПО как отдельным пользователям, так и целым корпорациям. Это обусловлено тем, что нельзя ожидать одновременного перехода всех на СПО. Какое-то время одновременно и параллельно будут использоваться как проприетарные, так и свободные программы в различных областях и в различных ведомствах. При этом необходимо будет также поддерживать обмен информацией между обеими группами программ. Этот период по времени

может быть довольно продолжительным. А искусственно сокращать его не всегда целесообразно, поскольку имеются уникальные проприетарные программы, которые не имеют свободных аналогов. Для решения же большинства встречающихся практических задач имеется масса свободных программ – необходимо лишь выбрать наиболее подходящую и освоить работу с ней, например, под управлением VirtualBox. В таком случае технический переход на СПО не вызовет никаких проблем.

СПИСОК ИСТОЧНИКОВ

1. Берлинер Э.М., Глазырина И.Б., Глазырин Б.Э. Windows XP. Самоучитель. – М.: ЗАО «Издательство БИНОМ», 2002 г. – 416 с,: ил.
2. Бэрри Мане. Компьютерные сети: Пер. с англ. – М.: БИНОМ, 1995. – 400 с.
3. Вин Дж. Искусство web-дизайна. Самоучитель. СПб.: Питер, 2002. – 224 с.: ил.
4. Гудман Д., Моррисон М. JavaScript. Библия пользователя, 5-е издание.: Пер. с англ. – М. : ООО “И.Д. Вильямс“, 2006. – 1184 с. : ил.
5. Далхаймер К., Уэлш М. Запускаем Linux, 5-е издание. – Пер. с англ. – СПб.: Символ Плюс, 2008. – 992 с.: ил.
6. Дронов В.А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. – СПб.: БХВ-Петербург, 2011. – 416 с.: ил.
7. Дунаев В. Самоучитель JavaScript, 2-е изд. – СПб.: Питер, 2005. – 395 с.: ил.
8. Дэйт К. Введение в системы баз данных. Пер. с англ. – М., Наука, 1980.
9. Евдокимов В.В. и др. Экономическая информатика. Учебник для вузов. Под ред. д.э.н. проф. В.В. Евдокимова. — СПб.: Питер, 1997.
10. Информатика для экономистов: Учебник / Под общ. ред. В.М. Матюшка. – М.: ИНФРА-М, 2007. – 880 с
11. Информатика для юристов и экономистов / Симонович С.В. и др. – СПб.: Питер, 2001. – 688 с.
12. Информатика и информационные технологии: учебное пособие / Ю.Д. Романова, И.Г. Лесничая, В.И. Шестаков, И.В. Миссинг, П.А. Музычкин; под ред. Ю.Д. Романовой. – 3-е изд., перераб. и доп. – М.: Эксмо, 2008. – 592 с. – (Высшее экономическое образование).
13. Информатика. Базовый курс / Под ред. С.В. Симоновича – СПб: Издательство «Питер», 2000. – 640 с: ил.
14. Информатика. Теория и практика: Учеб. Пособие / В.А. Острейковский, И.В. Полякова. – М.: Издательство Оникс, 2008. – 606 с.: ил.
15. Кирсанов Д. Веб-дизайн: книга Дмитрия Кирсанова. СПб: Символ-Плюс, 2006. – 376 с.: цв. ил.

16. Колисниченко Д.Н. Linux. От новичка к профессионалу. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2010. – 784 с.: ил.
17. Колисниченко Д.Н. Самоучитель Linux. Установка, настройка, использование. – 4-е изд., перераб. и доп. – СПб.: Наука и Техника, 2006. – 688 с: ил.
18. Колисниченко Д.Н. Самоучитель системного администратора Linux. – СПб.: БХВ-Петербург, 2011. – 544 с.: ил.
19. Коржинский С.Н. Настольная книга Web-мастера: эффективное применение HTML, CSS, JavaScript. Издание второе, исправленное и дополненное. – М.: Издательский торговый дом «КноРус», 2000. – 320 с.
20. Костромин В.А. Самоучитель Linux для пользователя. – СПб.: БХВ-Петербург, 2005. – 672 с.: ил.
21. Кролл Эд. Все об INTERNET: Руководство и каталог / Пер. с англ. С. М. Тимачева. – К.: Торгово-издательское бюро BHV, 1999. – 592 с.
22. Кузнецов М.В. MySQL 5 – СПб,: БХВ-Петербург, 2010. – 1024 с.: ил.
23. Леонтьев В.П. Новейшая энциклопедия Интернета 2006. – М.: ОЛМА-ПРЕСС Образование, 2006. – 720 с.: ил. – (Новейшая энциклопедия).
24. Лесничая И.Г., Миссинг И.В., Романова Ю.Д., Шестаков В.И. Информатика и информационные технологии. Учебное пособие / Под ред. Романовой Ю.Д. – М.: Изд-во Эксмо, 2005. – 544 с. – (Высшее экономическое образование)
25. Лоу Д. Компьютерные сети для «чайников». – К.: «Диалектика», 1995. – 256 с.
26. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
27. Мальчук Е.В. HTML и CSS. Самоучитель. – М.: Издательский дом «Вильямс», 2006. – 416 с.: ил.
28. Мартин Дж. Организация баз данных в вычислительных системах. М.: Мир, 1980.

29. Матросов А.А., Сергеев А.О., Чаунин М.П. HTML 4.0. – БХВ-Петербург, 2004. – 672 с.: ил.
30. Мюллер Дж. Оптимизация Windows XP. – СПб.: Питер, 2006. – 480 с.: ил.
31. Ногл М. TCP/IP. Иллюстрированный учебник – М.: ДМК Пресс, 2001. – 480 с.: ил.
32. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 3-е изд. – СПб.: Питер, 2007. – 958 с.: ил.
33. Основы информатики. Лекции по курсу Информатика. Для студентов первого и второго курсов. / А.Б. Костенко, Б.И. Погребняк, Н.В. Гринчак, Т.А. Холодная – Харьков: ХГАГХ, 1997. – 170 с.
34. Поляк-Брагинский А. В. Локальные сети. Модернизация и поиск неисправностей. – СПб.: БХВ-Петербург, 2006. – 640 с.: ил.
35. Попов Э.В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука. Гл. ред. Физ.-мат. Лит., 1987. – 288 с.
36. Ресиг Д. JavaScript. Профессиональные приемы программирования. – СПб.: Питер, 2009. – 720 с.: ил.
37. Ресиг Д., Бибо Б. Секреты JavaScript ниндзя: Пер. с англ. – М. : ООО “И.Д. Вильямс“, 2013. – 416 с. : ил.
38. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер. 2002. – 544 с.: ил.
39. Степанов А.Н. Информатика: Учебник для вузов. 4-е изд. – СПб.: Питер, 2006. – 684 с.: ил.
40. Таунсенд К., Фохт Д. Проектирование и программная реализация экспертных систем на персональных ЭВМ: Пер. с англ./ Предисл. Г.С. Осипова. – М.: Финансы и статистика, 1990. – 320 с.: ил.
41. Уотермен Д. Руководство по экспертным системам: Пер. с англ. – М.: Мир, 1989. – 388 с., ил.
42. Флэнаган Д. JavaScript. Подробное руководство. Пер. с англ. – Символ-Плюс, 2008. – 992 с., ил.

43. Экономическая информатика: / Под ред. П.В. Коноховского и Д.Н. Колесова. – СПб.: Питер, 2000. – 560 с.
44. Экономическая информатика: Учебник / Под ред. В.П. Косарева. – 2-е изд., перераб. и доп. – М.: Финансы и статистика, 2004. – 592 с.: ил.
45. HTML в примерах А. Климова. – [Электронный ресурс] – Режим доступа: http://winchanger.narod.ru/webmaster.html.
46. HTML справочник Владимира Городулина. – [Электронный ресурс] – Режим доступа: http://html.manual.ru/.
47. В 2014 году в Харькове откроют IT-парк. Официальный сайт Харьковского городского совета, городского головы, исполнительного комитета. 14/09/2012 – [Электронный ресурс] – Режим доступа: http://www.city.kharkov.ua/ru/news/u-2014-rotsi-v-harkovi-vidkriyut-it-park-15807.html.
48. Виртуальная среда обучения КНИТУ (КХТИ). – [Электронный ресурс] – Режим доступа: http://www.moodle.ipm.kstu.ru/.
49. Левшин И. ИТ-обеспечение выборов на Украине. «ComputerWorld Россия», № 42, 1999, [Электронный ресурс] – Режим доступа: http://www.osp.ru/cw/1999/42/38241/.
50. Национальный Открытый Университет «ИНТУИТ». – [Электронный ресурс] – Режим доступа: http://www.intuit.ru.
51. Поисковая система «Рамблер» . – [Электронный ресурс] – Режим доступа: www.rambler.ru/.
52. Постанова Кабінету Міністрів України «Про затвердження Державної цільової науково-технічної програми використання в органах державної влади програмного забезпечення з відкритим кодом на 2012-2015 роки» № 1269 від 30.11.2011 р. [Электронный ресурс] – Режим доступа: http://zakon4.rada.gov.ua/laws/show/1269-2011-%D0%BF.
53. Проект Закону про внесення змін до Закону України "Про Національну програму інформатизації" щодо використання відкритого програмного забезпечення в органах державної влади, органах місцевого

самоврядування, державних установах, державних підприємствах та навчальних закладах державної і комунальної форми власності. – [Электронный ресурс] – Режим доступа: http://w1.c1.rada.gov.ua/pls/zweb2/webproc34?id=&pf3511=42379&pf35401=212492.

54. Рекомендації парламентських слухань на тему «Законодавче забезпечення розвитку інформаційного суспільства в Україні» схвалені Постановою Верховної Ради України № 1565-VII від 03.07.2014 р. – [Электронный ресурс] – Режим доступа: http://zakon4.rada.gov.ua/laws/show/1565-18#n12.
55. Розпорядження Кабінету Міністрів України «Про схвалення Концепції розвитку електронного урядування в Україні» №2250-р від 13.12.2010 – [Электронный ресурс] – Режим доступа: http://zakon1.rada.gov.ua/cgi-bin/laws/main.cgi?nreg=2250-2010-%F0.
56. Современный учебник JavaScript. – [Электронный ресурс] – Режим доступа: http://learn.javascript.ru/.Союз образовательных сайтов. – [Электронный ресурс] – Режим доступа: http://allbest.ru/union/.
58. Украина вошла в пятерку стран, предоставляющих фриланс-работников. Дело от 19 октября 2012 г. – [Электронный ресурс] – Режим доступа: http://delo.ua/business/ukraina-voshla-v-pjaterku-stran-predostavljajuschih-frilans-rabotniko-187652/.
59. Учебник по Html (хтмл) для чайников Алленовой Натальи. – [Электронный ресурс] – Режим доступа: http://postroika.ru/html/.
60. Учебные материалы для студентов. – [Электронный ресурс] – Режим доступа: http://www.4stud.info/.
61. Центр дистанционного обучения Харьковского национального университета городского хозяйства им. А. М. Бекетова. – [Электронный ресурс] – Режим доступа: http://cdo.kname.edu.ua/.

Навчальне видання

ПОГРЕБНЯК Борис Іванович,
КОСТЕНКО Олександр Борисович,
БУЛАЕНКО Марина Володимирівна

ІНФОРМАТИКА

КОНСПЕКТ ЛЕКЦІЙ

(для студентів 1-го курсу денної та заочної форм навчання освітньо-кваліфікаційного рівня бакалавр за напрямами підготовки 6.030504 – Економіка підприємств і 6.030509 – Облік і аудит)

(рос. мовою)

Відповідальний за випуск *М.І. Самойленко*

За авторською редакцією

Комп'ютерне верстання: *Б. І. Погребняк*

План 2015, поз. 158 Л

Підп. до друку 19.03.2015 Формат 60х84/16
Друк на ризографі Ум. друк. арк. 110,880
Тираж 50 пр. Зам. №

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Революції, 12, Харків, 61002
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК №4705 від 28.03.2014 р.